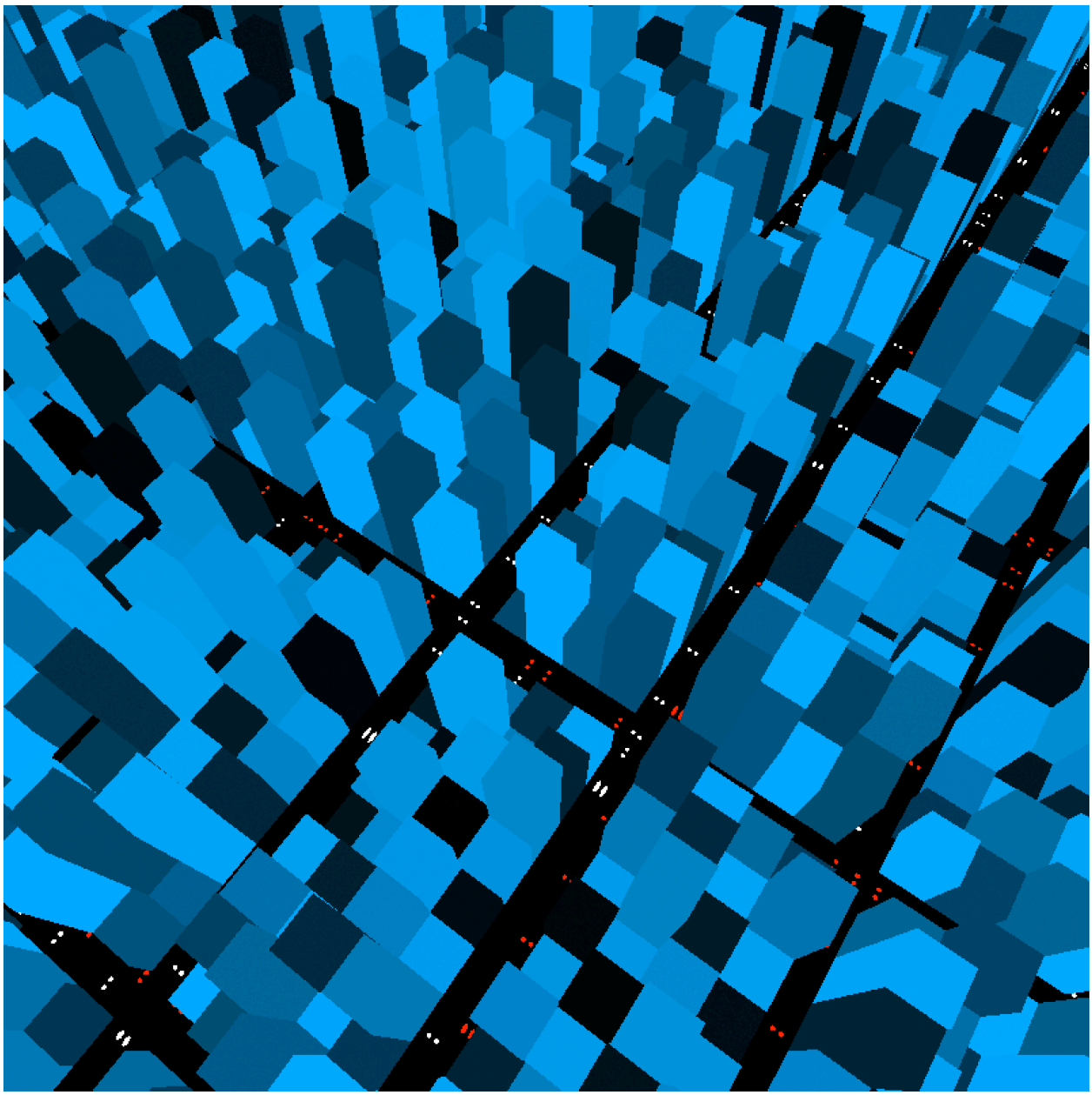


Use of Cellular Automata in Creating a City Simulation

Dmitry Konev

Sarah Lawrence College

The goal for this project is to create a cellular automaton rule set in two dimensions that can be used to approximate traffic patterns in a place such as Manhattan. This can be used for many purposes, including experimenting with finding a sustainable level of traffic density, or using this simulation as part of an art project.

A cellular automaton is a model that consists of a grid of cells, each of which have a set of possible states that changes after each step depending on the cell's previous state and the state of its nearby cells (its neighborhood).

For the sake of explaining how this model would relate to actual vehicles, the rule can be described as if cars are objects that move from one cell to the next. This is not actually the case in a cellular automaton. What actually happens is that each cell's value is determined by the state of the cell and its neighbors in the last "generation". So, what might be described as a "car moving to the right" is actually two separate actions: "occupied cell with an empty neighbor to the right becoming empty", and "empty cell with an occupied neighbor to the left becoming occupied".

I. Creating the Cellular Automaton ruleset

We started building the model from a cellular automaton called Rule 184, also sometimes called "traffic rule" due to its similarity to car traffic patterns: This rule can be used as a model to describe multiple cars moving to the right, in which a car can only move if the next cell is empty. Rule 184's ruleset is:

- If a cell is occupied, and the next (right) cell is empty, the cell becomes empty
- If a cell is empty, and the previous (left) cell is occupied, the cell becomes occupied
- If a cell is occupied, and the next cell is occupied, the cell stays occupied
- If a cell is empty, and the previous cell is empty, the cell stays empty

Rule 184 cells may only have two different states:

- Occupied
- Empty

A new rule that we will call **Beyond 184** was created by modifying the original Rule 184. The ruleset was expanded for the sake of creating a more "interesting" simulation – one that would more accurately represent traffic (by allowing cars to move at different speeds) and would be more aesthetically pleasing when used as part of an artistic project. In this expanded rule, occupied cells have "cooldown" and a "wait timer". The cooldown defines how long a "car" will wait in place before moving to the right, and the actual timer counts down as generations progress.

If we define a single cell to be equivalent to a 10 m^2 square in the real world, and one generation to be a second of real time, there's a rough equivalence between how cells "move" and how cars travel in the real world:

Cooldown	Speed	Notes
0	10 m/s (22 mph)	Roughly equivalent to Manhattan speed limit
1	5 m/s (11 mph)	
2	3.33 m/s (7 mph)	

Going further with this idea, it was made so that vehicles moving at higher speeds cannot stop immediately. Using the brakingdistances.com calculator, we can see the stopping distance at various speeds:

Cooldown	Speed	Stopping distance	Cells travelled
0	10 m/s (22 mph)	14 m	~2 cells
1	5 m/s (11 mph)	5 m	~1 cell
2	3.33 m/s (7 mph)	3 m	0 cells

Therefore, a cell with **cooldown 0** will behave just like a cell in the standard rule 184. It will immediately attempt to move to the right, with an empty cell to the right of it becoming occupied in the next generation, and this specific cell becoming empty. A cell with **cooldown 1 or 2** will "sleep" for 1 or 2 generations respectively – that is, the cell to the right will *not* become occupied if the cell to the left has a cooldown with a wait timer that has not reached zero.

Having a lower cooldown comes with a drawback – a "car" that has **cooldown 0 or 1** will not be able to come to a stop, even if the cell immediately to the right is occupied. To compensate for this, cells will "slow down" if any of the cell's 4 neighbors to the right are occupied.

This also means that if there is a situation where a "car" with **cooldown 0** has a neighbor that also has a "car" to the right of it, there will be a "car wreck". While this behavior will not appear naturally in a 1-D simulation, as a situation where a "car" will not be able to "see" an incoming obstacle cannot occur (that is, there are no turns that can obscure the "car's" ability to look at its neighbors to the right, so it will be able to brake in time in all cases, unless the cells to the right of it are manually changed), it will become useful when this model is moved to the two-dimensional world.

The following page includes the ruleset for this expanded rule, and a list of all possible states.

Beyond 184 ruleset:

- If a cell is empty, and the cell to the left is not occupied by a “car” type with a wait timer value of 0, the cell stays empty.
- If a cell is empty, the cell to the left is occupied by a “car” type that has a wait timer value of 0, or if a cell is occupied by a “car” type, the wait timer is 0, the cell to the right is empty, and the cell to the left is occupied by a by a “car” type, with wait timer 0, and cooldown not 2
    - …and any of the four cells to the right are occupied, then the cell becomes occupied by a “car” type, the “cooldown” value is increased by 1, unless the cell’s current cooldown value is 2, in which case it stays at 2, and the cell’s “wait timer” value is set to be equal to the “cooldown” value.
    - …and all of the four cells to the right are empty, then the cell becomes occupied with a “car” type, the “cooldown” value is decreased by 1, unless the cell’s current cooldown value is 0, in which case it stays at 0, and the cell’s “wait timer” value is set to be equal to the “cooldown” value.
- If a cell is occupied by a “wreck” type, and the wait timer is not 0, the timer is decremented by 1.
- If a cell is occupied by a “wreck” type, and the wait timer is 0, the cell becomes an empty cell.
- If a cell is occupied by a “car” type, and the wait timer is not 0, the timer is decremented by 1.
- If a cell is occupied by a “car” type, the wait timer is 0, and the cooldown is not 2, the cell becomes empty.
- If a cell is occupied by a “car” type, the wait timer is 0, the cooldown is 2, and the cell to the right is empty, the cell becomes empty.
- If a cell is occupied by a “car” type, the wait timer is 0, the cooldown is 2, and the cell to the right is not empty, the cell stays as a cell with a “car” type, cooldown 2, and wait 0.

Beyond 184 possible cell states:

- Empty
- Car with cooldown 2 and wait timer 2
- Car with cooldown 2 and wait timer 1
- Car with cooldown 2 and wait timer 0
- Car with cooldown 1 and wait timer 1
- Car with cooldown 1 and wait timer 0
- Car with cooldown 0 and wait timer 0
- Wreck with wait timer 9
- Wreck with wait timer 8
- Wreck with wait timer 7
- Wreck with wait timer 6
- Wreck with wait timer 5
- Wreck with wait timer 4
- Wreck with wait timer 3
- Wreck with wait timer 2
- Wreck with wait timer 1
- Wreck with wait timer 0

II. Bringing the Beyond 184 ruleset to two dimensions

Working in one dimension is quite limiting. It is impossible to simulate how cars behave at intersections and turns, for example. Therefore, the simulation needs to be converted to work in two dimensions, creating a model that looks somewhat like a map.

Both the Rule 184 and the Beyond 184 cellular automata can be adapted to work in two dimensions. The way that we have chosen to do this involves adding a "direction" attribute to cells. This attribute dictates what cells will be a part of that cell's "neighborhood". There are four possible directions – north, east, south, and west.

A cell with its direction set to "east" will behave the same way as it does in the one-dimensional automata – checking its neighbors to the left (previous cell) and right (next cell). A cell with its direction set to east will be "mirrored" compared to that, with the rule being seemingly reversed – that is, this cell will check its right neighbor to decide if it becomes occupied or not (the right cell will be the previous cell), and check its left neighbors to decide if it becomes empty or not (the left cell will be the next cell).

↓	←	←	←
↓	↑	→	→
↓	↑		
↓	↑		

Example of a right turn in a 2-lane road

By using this way of creating the two-dimensional automata, it is possible to use the rules already implemented with minimal modifications.

The modifications made, however, involved adding a new possible type for cells, called road, and a new attribute, called direction, as described earlier. This is necessary to allow cells to distinguish between places where it is and is not possible to drive.

Upon implementing these attributes, while the simulation was indeed running in two dimensions, there were some issues.

The first issue is "ghost" cars, as seen below using a simplified example, with cells that have a dark background representing cells with cars:

↓	←	←	←
↓	↑	→	→
↓	↑		
↓	↑		

Generation 1

↓	←	←	←
↓	↑	→	→
↓	↑		
↓	↑		

Generation 2

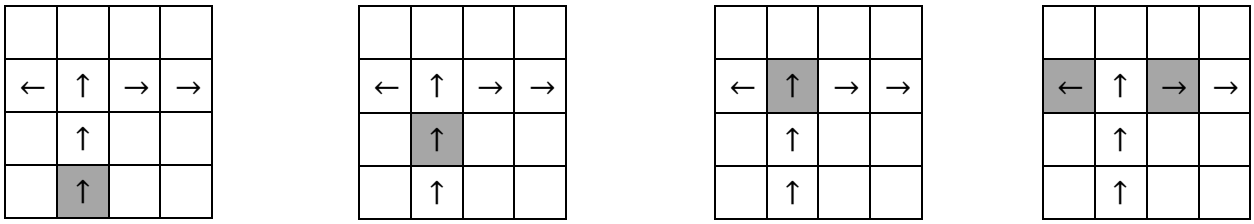
↓	←	←	←
↓	↑	→	→
↓	↑		
↓	↑		

Generation 3

↓	←	←	←
↓	↑	→	→
↓	↑		
↓	↑		

Generation 4

The issue appears in the cell with the dashed line. Because the cell's direction is "north", and the cell above it is occupied, the dashed line cell, following its rule, is not able to become an empty cell. The cell to the right, however, following the same rule, but with direction set to "east", becomes occupied. Therefore, a situation occurs where a cell does not become empty when it should.
The second issue is "car duplication", as demonstrated below:

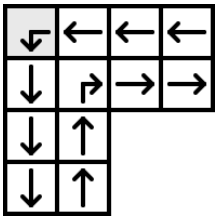


Generation 1 *Generation 2* *Generation 3* *Generation 4*

In this case, a situation occurs where the same occupied cell is being used as the "source" for multiple empty cells. This means that one "car" somehow turns into two cars, which is not a desirable behavior.

To fix this, the "direction" attribute was broken up into two parts – "source" and "destination". "Source" dictates what cell will be considered the previous cell, and destination dictates what cell will be considered the next cell. This will solve the "ghost car" problem by making that cell look at the correct neighbor.

By adding an additional rule that requires the source of the current cell to match with the destination of the previous cell, and the destination of the current cell to match with the source of the next cell, we can solve the "car duplication" problem, and also create the necessary component for adding intersections (the ability to choose which direction a "car" will turn in). In the above example, we can regulate the direction the car goes in (west or east) by changing the center cell's "destination" attribute.



A turn in the final implementation

To finish the transition to two dimensions, it is also necessary to handle intersections.

Intersections can have quite complex configurations. For example, when options such as the ability for vehicles to turn left in an intersection between two two-lane roads are considered, an intersection may have around six possible configurations, to account not just for allowing traffic to pass for both roads, but to also allow vehicles to make turns from one road to another – since it will be required to also temporarily stop traffic on one side of the road to allow for left turns.

While it is possible to implement such an intersection with traffic lights in this simulation, it would involve adding a large amount of complexity. Particularly, in an intersection between two two-lane roads, a 2x2 block of cells will be required to act as a single entity when determining which direction vehicles are allowed to turn in. While this may not necessarily be against the definition of cellular automata, and may be implemented in a way that would not require a rule to operate on multiple cells at once (for example, by creating a signal cell that other cells can check independently), it would go against general conventions that are followed by most cellular automata rulesets, which all specifically have rules that only modify one cell at a time, based on the state of its neighbors in a previous generation.

A good alternative to typical intersections was suggested: roundabouts. According to the Federal Highway Administration, roundabouts are more efficient, cheaper, and reduce serious accidents by 80%, compared to typical intersections. Roundabouts were previously used for such a model by B. Chopard, P. O. Luthi and P-A Queloz in their paper “Cellular automata model of car traffic in a two-dimensional street network”, though implemented in a different manner from what our model will implement.

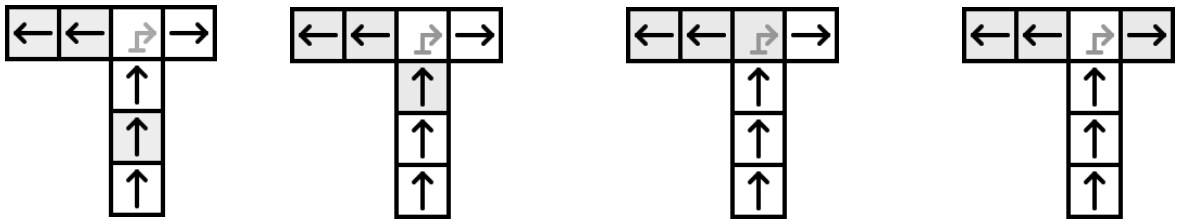
Roundabouts do not use traffic lights, and vehicles may freely enter them at any point in time. For the simulation, this means that implementing roundabouts will not require a complex arrangement of cells to stay in sync – instead, they will simply need to change their “source” and “destination” parameters to allow for vehicles to enter, exit, or move through the roundabout.

For the purpose of implementing such a roundabout, an additional road type was added, called “intersection”. This road type includes two additional attributes called “iSource” and “iDestination”, which determine the alternative source and direction attributes that the cell may switch to.

Additional logic was also implemented to determine wherever the source and destination attributes should either remain as they are defined originally, or be swapped with their iSource and iDestination counterparts.

For the destination attribute, both possible destinations are checked to see which one has less traffic. To do so, an attempt is made to follow the path (that is, an unbroken chain of cells where the next cell’s “source” attribute matches the previous cell’s “destination” attribute) that both destination attributes point to for five cells, then the destination that has less occupied cells in the path that it points to is chosen. If both destinations point to paths that have the same number of occupied cells, then the destination will alternate each generation.

Demonstrated below is a simple intersection, where the intersection cell is the cell with an arrow that has a tail and is colored by a lighter shade of gray:

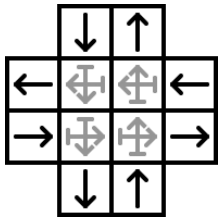


Generation 1 *Generation 2* *Generation 3* *Generation 4*

In this example, the intersection cell’s destination always points to the right side, which points to a path that is empty, instead of pointing to the left side, which points to a path that has two occupied cells.
For the source parameter, the process is very similar, except the path chain is followed backwards, and the source direction that points to the road with *more* vehicles is chosen.

Implementing the intersection cell logic in this manner adds a degree of traffic control: intersection cells always strive to redirect traffic from busier paths to less busy paths. By attempting to follow this chain of road cells, as opposed by simply checking the next five cells in the given destination (similar to how the rule for car cells works in the Beyond 184 ruleset), the intersection cells gain the ability to properly check roads that, for example, may have turns in them. This naturally adds a similarity to the real world: an actual intersection may have various sensors installed that influence which directions have higher priority, but drivers will not have the ability to see vehicles that are around the corner.

Using this new intersection cell, it is actually possible to compress the roundabout design to occupy the same 2x2 space that the hypothetical complex intersection with traffic lights would have taken. Pictured on the right, this more portable roundabout's cells make use of the ability to change both source and destination parameters.



2x2 roundabout

Finally, some changes were made to the Beyond 184 ruleset to work better with intersections:

A change was made to make cars slow down (that is, move to a higher cooldown) when they come near an intersection. This was implemented by treating intersection cells as "occupied" cells when the cell checks its extended neighborhood for obstacles to determine if it should speed up or not.

Another change was made to prevent cars from accelerating while currently being on intersection cells.

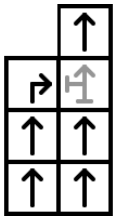
Last, to prevent cars going at higher speeds from disappearing in situations where they will not be able to come to a stop before the intersection that has a source direction that does not match the car's current cell's destination, intersection cells will also check the currently inactive source direction for any incoming cars – and if it is a car that will not be able to stop, they will become occupied by a car. If such a situation occurs where cars from both possible source directions attempt to occupy the same intersection cell, this intersection cell will instead become occupied by a wreck cell.

III. Use of this simulation

While this project's main goal was to create a reasonably realistic traffic simulation for use in an artistic programming project, there are many other scenarios where it can be applied, the biggest one being traffic modeling. Compared to a standard Rule 184 simulation, a simulation with the Beyond 184 ruleset will offer more information, which would allow to explore trends such as the average speed of vehicles in a city. It will also simulate more realistic vehicle behaviors, such as braking times at various speeds, and even car crashes. This can help with creating a more nuanced model that can not only assist in exploring different traffic density levels, but also help discover how fast vehicles

can move depending on how much traffic there is, and what impact things like sharp turns may have on road safety.

The road-intersection cell structure taken for bringing the Beyond 184 ruleset to two dimensions allows for simulating various complex street rules. The intersection cells can be used for more than just simulating their namesake or roundabouts. For example, as seen to the right, they can be used to implement road merges. It is also possible to use them to create highways where vehicles have the ability to overtake one another.



Road merge

In addition to that, this structure allows the road-intersection model to be used to adapt various other 1-D cellular automata to work in two dimensions. For example, due to the fact that all the changes made to the Beyond 184 ruleset made to implement intersections all involved ensuring that the car cells' extended neighborhood interactions all worked correctly, such changes would not be necessary when adapting the standard Rule 184 to work in two dimensions, as that ruleset only requires interacting with the cell's immediate neighbors.

Finally, insight can be gained by modifying how the model works. By modifying other parameters, such as how far along do intersection cells try to follow roads, or by changing the underlying traffic simulation ruleset, the differences in behavior of the model can be applied to actual traffic laws.

For example, during the process of implementing intersections, the rule that makes cars slow down near an intersection cell made a big impact: the model went from having a traffic accident at a roundabout approximately every 600 generations, to running without any such accidents for a million generations after the rule was implemented. Implementing this rule had a similar impact to curvature at actual roundabouts: they force vehicles to move at slower speeds while inside the roundabout, which allows them to come to a full stop faster.

The source code for the implementation of the Beyond 184 ruleset will be made available online for anyone to explore and modify, and will hopefully be useful in future research involving the Rule 184 cellular automata, traffic modeling, and the process of bringing 1-D cellular automata to two dimensions.

Works Cited

"Intersection Safety – Safety." *Federal Highway Administration*, safety.fhwa.dot.gov/intersection/innovative/roundabouts/.

"Calculate Braking and Stopping Distances." *Calculate Braking and Stopping Distances*, brakingdistances.com/.

"Cellular Automaton." *Wolfram MathWorld*, mathworld.wolfram.com/CellularAutomaton.html.

Chopard, B, et al. "Cellular automata model of car traffic in a two-Dimensional street network." *Journal of Physics A: Mathematical and General*, vol. 29, no. 10, 1996, pp. 2325–2336., doi:10.1088/0305-4470/29/10/012.