Pedestrian Traffic Analysis of Trindade's Station

Eduardo Correia
Faculty of Engineering of University of Porto
up201806433@up.pt

Henrique Sousa
Faculty of Engineering of University of Porto
up201906681@up.pt

Mª Francisca Almeida
Faculty of Engineering of University of Porto
up201806398@up.pt

***Abstract*—Rush hour embarking and debarking at a transit platform can cause the intersection of large masses of people, making it harder for easy movement and arriving at the destination. A good design of a multi-modal transit station might help to attenuate this problem, bringing improvements in metrics such as mean average time to changing lines and, more subjectively, stress induced in passengers. We develop and describe a simulation in Netlogo that is tested in the Trindade, Porto, multi-modal transit station. Multiple scenarios are considered to exemplify its usefulness in evaluating changes to the infrastructure.**

***Index Terms*—modeling, system modeling, pedestrian traffic**

I. Introduction

Public transportation represents one of the most efficient ways to move large masses of people. Large multi-modal transit stations, where multiple modes of transportation coexist, have a lot of pedestrian traffic. Rush hour embarking and debarking between platforms, stops, or bays has large masses of people crossing each other and navigating the station space, often in directly opposite directions. It is in this overall context that a station with multiple floors serving multiple train platforms becomes an interesting problem, for it poses optimization problems not only in scheduling but also in the design of the station itself for faster traveling and cheaper maintenance and construction.

In the following work, we attempt to create a simulation of that scenario. We emphasize developing natural path-finding algorithms, particularly across multiple floors, in a computationally efficient way. Collisions between passengers are also taken into account. The developed model also attempts to generalize as much as possible, allowing for different scenarios to be tested easily and changing of parameters.

In section II, we mention some projects related to the pedestrian traffic problem, particularly in subway stations. In section III, we outline the main implementation details and thought process. In section IV, we analyze multiple simulated scenarios by changing the environment and the running parameters. Finally, in section V, we lay out the conclusions of the work and future work.

II. Related work

With the increasing demand for public transportation, as city centers grow in population, subway stations have to adapt to accommodate the flow of all the passengers with minimal impediments.

As such, several projects exist regarding the simulation of pedestrian flow in stations as a means to analyze different phenomena and alternative scenarios before carrying out any changes to existing stations.

Some examples of successful stories include the Plaça de Catalunya Station [3], which was remodeled according to a chosen design out of four after using the simulation software LEGION to analyze the most critical metrics.

In the same fashion, a similar project was conducted for Moscow's metro [1], the busiest in Europe. A model was developed that reflected

passenger flow from disembarking from one platform to another and calibrated using real-life data. Ultimately, a proposed solution to change the station topology was accepted.

Examples like these show how useful and insightful simulation and modeling are for optimizing traffic flow in railway stations. Their success boosts the validation of our project, and as such will be used as role models for our project.

III. Materials & Methods

A. Problem formalization

The transit station will be modeled with the following entities:

- **Floors**, each being a grid with the same size
- **Platforms**, part of a floor
- **Passengers**
- **Points of interest**, either a train or the outside
- **Portals**, either a stair (in practice, representing the top of the stairs) or an elevator
- **Obstacles**, representing walls, train tracks, or other fixtures (benches, trash bins, card validators).
- **Escalators**, either upwards or downwards

A floor is made up of one or more platforms. A platform is such that a passenger can get to any place on the platform directly, and other platforms are only accessible by changing floors (taking a portal).

Every entity is on a floor at any given point. Passengers can move to other floors by using portals. Different portals can have different behaviors.

A passenger spawns in a cell of a point of interest: its source. A passenger has as a destination: a cell of another point of interest. A passenger has a given radius and speed. Two passengers can collide with each other if they are too close.

If two portals of the same type are linked, they allow passengers to transition between the respective floors.

Using stairs, passengers transition between floors instantly. Using elevators, passengers transition between floors in a time proportional to the number of floors traveled. For simplicity, elevators are static and not dynamic entities. As such, there is no capacity and no floor where the elevator is. When reaching an elevator, a passenger can use it as if it were their own personal elevator.

Passengers cannot move through or be too close to obstacles.

Every part of a floor that isn't an escalator or an obstacle is deemed as *normal ground*.

Escalators provide an increase in passenger speed. An escalator has three logical parts: entry, body, and exit. A passenger can only move to the entry from the normal ground or any escalator's exit and, from there, can only move to the body. A passenger can only move to the body from the entry and, from there, can only move to the exit. A passenger can only move to the exit from the body and, from there, can go to normal ground or any escalator's entry. We will call *moving restrictions* the conditional movement set by escalators.

Every entity but the passenger is considered as being the environment. Points of interest and portals are henceforth referred to as static entities. Static entities are made up of several cells. A static entity has a centroid, which is one of its cells. The centroid is the patch used for all of the path-finding operations (see subsection III-E)

B. Materials

The described model was developed in the context of the simulation tools used for this work: NetLogo [2].

Additionally, the Trindade station, the main hub of the subway system of the city of Porto, Portugal, was modeled. Two pictures from its emergency evacuation plan and data collected from Google Maps were used to get a to-scale color-coded image representation of the station to be fed as the simulation's environment. Each color represents one of the aforementioned environmental entities.

C. Pipeline

The simulation happens as follows:

1) Prepare the environment
   a) Loading of the image. The colors are transformed into variables and data structures within the simulation.
   b) For every platform, initialize the path-finding between static entities
      i) Link corresponding portals
      ii) Path-finding is done bidirectionally between every distinct pair of the aggregate of points of interest and portals (Equation 1)
2) Prepare the simulation run
   a) Spawn the passengers one by one
      i) Spawn the passenger in a random cell from all of the cells of the points of interest
      ii) Choose the destination from the cells of all of the other points of interest
      iii) Calculate the shortest path to be taken by the passenger through the different floors.
3) Run the simulation
   a) For all passengers
      i) Move the passengers to the next point of their path
      ii) Collide the passengers between each other
      iii) Repel passengers from walls
      iv) Log data
      v) Update passengers' path
4) Save data to a file

$$\forall a, b \; pathfind\,(a, b) : a, b \in (PoI \bigcup Portals)\,, a \neq b \tag{1}$$

D. Platforms and floors

A transit station is often made up of multiple floors. We didn't use the 3D version of NetLogo for the work as the environment isn't truly three-dimensional but could instead be considered as multiple, discrete, connected 2D layers. Therefore, the image representation of the station is a series of same-sized rectangles of the subsequent floors. Figure 1 represents an example of a station with three floors and four platforms.

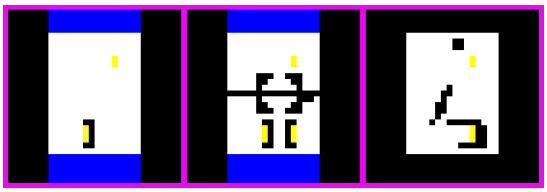

Fig. 1. Image representation of a station with three floors with obstacles (black), stairs (yellow), and train tracks (blue). The floors are separated by boundaries (pink). The middle floor has two platforms

E. Static path-finding

Static path-finding is done for each platform for the centroids of the static entities that belong to it and between the portals that connect platforms. In the end, there are a series of complete bidirectional graphs between the static entities of each and every platform and bidirectional edges connecting those.

What we describe as path-finding for the portals is as simple as comparing the coordinates of the portal within a floor and matching it to others whose coordinates are the same. We didn't consider the possibility of a transit station with more floors that might have, for example, stairs in the same position but only connecting floors 1 to 2 and 3 to 4, but not 2 to 3. Floor adjacency could be introduced to tackle this issue.

Path-finding for the static entities is the conventional space traversal, which we will now analyze in depth.

In NetLogo, there is both a cellular automaton and Euclidean space approach. There are patches, which occupy a quadrangular space with a side of one unit, and turtles that can have *(x,y)* real coordinates that are, nonetheless, heavily linked to the patch (or cell) they are in.
Given that context, using a shortest-path algorithm for a grid feels intuitive. Our solution for path-finding does base itself on the *A** algorithm, using the Cartesian distance to the goal. However, to increase the simulation's realism, passengers must be able to move in any direction (not just in the eight directions of adjacent patches). Furthermore, we wanted to simplify itself so that

moving sequentially in the same direction could be stored as just one instruction. To achieve both these goals, we developed an algorithm to straighten the path.

First, the shortest path between two points is calculated. For each patch, adjacency is as defined by Von Neumann, only the four patches that share an edge.
Next, the path will be straightened. Being straight means that there are the passengers will not collide against any obstacle in the way and obey moving restrictions.
To do so, the path is iteratively straightened until the length of the path doesn't change. Let's call each iteration a *straightening step*. The straightening step is akin to an expanding window, where the window represents a sub-path we know is straight. Once the path is not straight, the previous window is saved, and the next expanding window can start on the position that broke the *straightness*.
The full algorithm is represented in pseudo-code in 1. Whilst not being perfect, the algorithm produces good results nonetheless.

Fig. 2. Two pairs of straightened (left) and not straightened-paths (right)

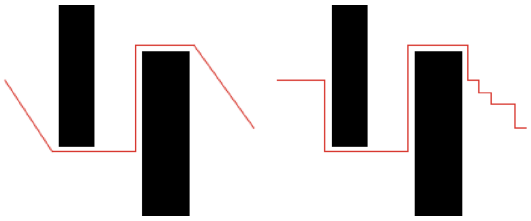

Fig. 3. A straightened (left) and not straightened path (right) around obstacles

As you can see, in Figure 3, the straightened path is not optimal. It is impossible to know a

Algorithm 1 Straigthen path algorithm

function STRAIGHTEN($path$)
  $path' \leftarrow StraightenStep(path)$
  $path'' \leftarrow StraightenStep(path')$
  while $len(path'' < path')$ **do**
    $path' \leftarrow StraightenStep(path)$
    $path'' \leftarrow StraightenStep(path')$
  end while
  return $path'$
end function

function STRAIGHTENSTEP($path$)
  $p \leftarrow 0$ ▷ Expanding window fixed pivot
  $i \leftarrow 2$ ▷ Expanding window moving part
  $newPath \leftarrow [\,]$
  while $i < len(path)$ **do**
    if not $IsStraight(path[p], path[i])$ **then**
      ▷ Store previous expanding window is stored, and a new one is set up
      ▷ Store previous straight sub-path
      $newPath \leftarrow newPath \vee path[i-1]$
      ▷ Pivot of the new window
      $p \leftarrow i - 1$
    end if
    ▷ Always add the last element
    $newPath \leftarrow newPath \vee path[-1]$
    $i \leftarrow i + 1$
  end while
  return $newPath$
end function

priori if a path will be optimized or not. One could theorize an approach that leverages the one used above to improve it further. At the time, the only solution we envisioned would be to brute force the set of all straightenings one could do by straightening in a different order than that shown in 1. Alternatively, a completely different method of path-finding that eliminates the need for straightening could have been developed. In $A*$, instead of adjacency being only the four patches with which an edge is shared, it could be every patch that forms a straight path from the given patch. This approach was ultimately not adopted out of fear of being too computationally heavy to calculate the neighbors.

For the straightening step, the function *Is-Straight* is used. It is paramount, for it is what determines if a certain path (or sub-path) is straight. It takes into account the size of the passenger. The algorithm's intuition is that the passengers' path (and its volume) is approximated by a rectangle (in whatever orientation) and intersected with any possible obstacles that the passenger would collide with. Implementation-wise, the start and end of the path are checked for collisions independently. The rectangle intersection is, in turn, done by *stepping* from the start to the end. The size of the step is linked to the precision of this approach: a bigger step means a smaller precision. In each step, a set of points is verified for possible collisions.

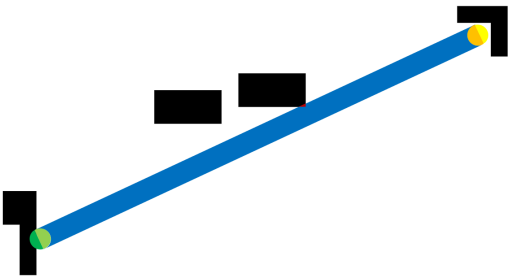

Fig. 4. Illustration of Is-Straight algorithm. Obstacles (black), passenger at start (green/light green circle), passenger at end (yellow/orange circle), passenger's path (blue rectangle), representation of collision (red), overlap of start and end with passenger's path (lime and orange semicircle, respectively)

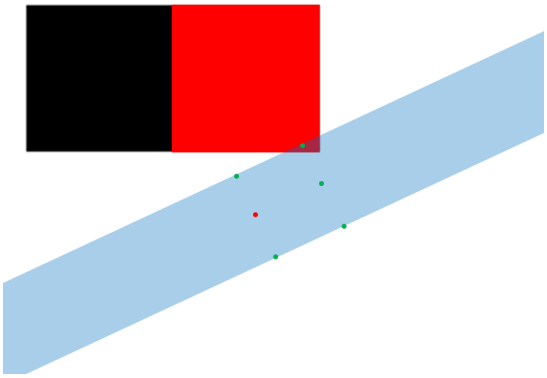

Fig. 5. Illustration of a step of Is-Straight algorithm. Obstacle (black), passenger's path (light blue), passenger's center in i-th step (red circle), passenger's collision verification points in i-th step (green circles), path with collision (red square)

In Figure 4, a representation of the algorithm can be seen. The intersection of the passenger at the end of the path with an obstacle would mean the path is not straight. If that were not the case, the path's intersection with an obstacle would also yield the same result. This is achieved as is seen in Figure 5. Since one of the collision-verified points lies within the obstacle (a straightforward operation in NetLogo), there is indeed a collision. Note that, with a different step, there is a chance the collision is not detected.

F. Passengers' path-finding

To improve the scalability of a run by allowing a great number of passengers, passengers' path-finding leverages the preexisting static path-finding.

Given the fact that a passenger most often wants to transition between platforms, it would be impossible for a naive path-finding algorithm in a 2d environment to work, given there are no patches that are neighbors with other floors and different platforms are physically separated from each other. To use said approach, interpreting the portals as patches that neighbor other floors would allow for the path-finding to proceed. However, the heuristic functions for $A*$ would have to take floors into account, making it not as trivial, and path-finding in the new floor would increase the size of the path-finding. The solution adopted is to separate the path-finding

on a platform and between platforms. To find the shortest path that traverses multiple platforms, we first calculate the remaining platform paths and use that to find the path between the platforms.

The remaining platform paths consist of, for each passenger, path-finding from the passenger's source to the portals in the source's platform and from the passenger's destination to the portals in the destination's platform. To do this, instead of rerunning the path-finding algorithm described in subsection III-E, which would be time-consuming for a large number of passengers, we are going to reuse the already found paths and try to adapt them to the position of the passenger.

In the case of the path between the source and one of the portals, we consider the path already found between the centroid of the point of interest the source belongs to (remember, a source is a cell of a point of interest and not its centroid) and the portal. This path is an ordered list of positions. From the first to the second position, there is a straight path; the second position is where a "bend" is located, followed by another straight segment to the third position. We assume that the distance from the cell to the centroid of the point of interest is not significant and, therefore, said bend would still apply would we rerun the path-finding algorithm. Therefore, we attempt to fit the already-found path to the new cell. To do this, we verify if the path from the source to the second position is a straight path, using the *Is-Straight* described in subsection III-E. If so, then the new path has been found. If not, then we choose the next point in the segment from the second position to the centroid of the point of interest until a path from the source to the chosen point is straight. Analogously for the destination.

In Figure 6 and Figure 7, two examples of this path merging can be seen. In Figure 7, as there wasn't a straight path from the source to the second point, the merge segment had to be calculated and doesn't connect straight to the second point.

For pathfinding between platforms, we use *Dijkstra's* pathfinding algorithm, not on the environment's grid, but on a graph with certain entities as nodes. Those entities will be the source and destination cells for the passenger and every portal in the whole environment. Below there is a figure illustrating the final graph

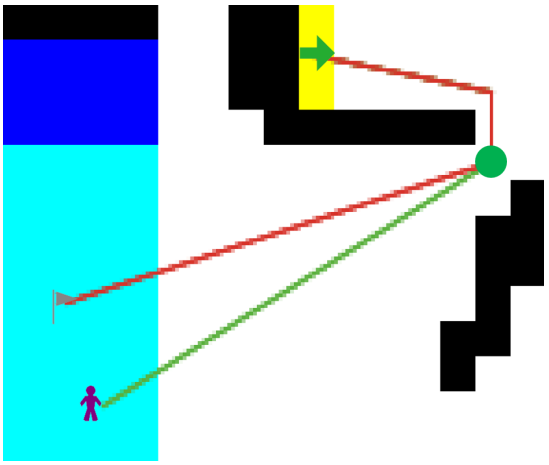

Fig. 6. Merging paths. The pre-calculated path (red line), new segment that is merged (green line), second point (green circle), point of interest centroid (gray flag), and passenger (purple person)

G. Collisions

The collision between the passengers is done to minimize the superposition of passengers. It uses the Lennard-Jones potential described by the equation

$$V_{LJ}(r) = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right]$$

IV. Results & Discussion

To test the devised simulation, we ran a series of simulations in Trindade, a major station in Porto's metro system (Portugal). Figure 9

We ran the following scenarios:

- Normal
- No elevators
- Only one set of stairs per train
- Longer trains
- With escalators

For each scenario, we ran with a passenger load of 150 and 300. The metrics collected are listed below. In section VI all of the tables with the metrics by experiment are listed. For each metric, a simple statistical analysis was performed.

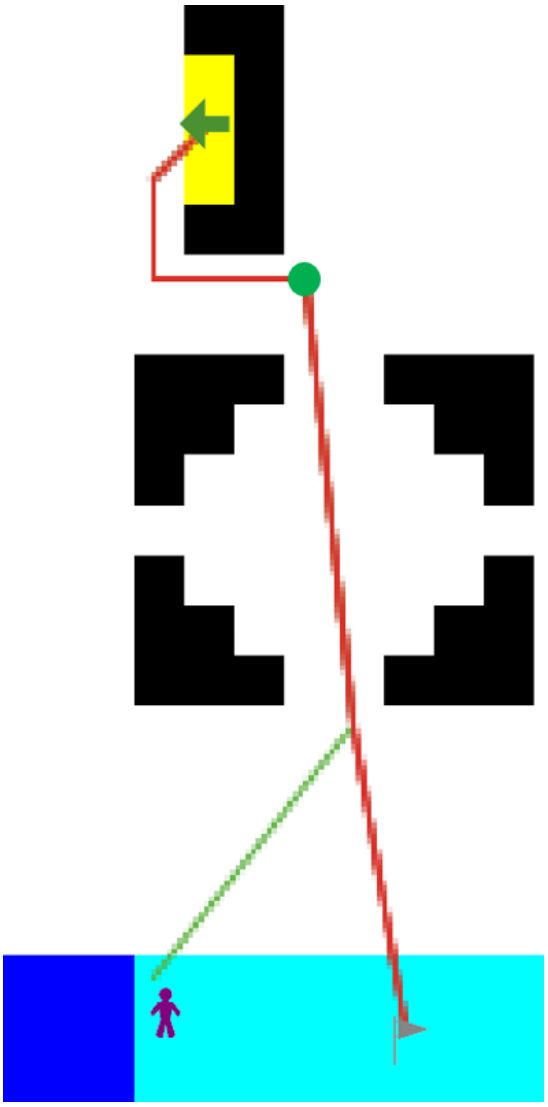

Fig. 7. Merging paths. The pre-calculated path (red line), new segment that is merged (green line), second point (green circle), point of interest centroid (gray flag), and passenger (purple person)

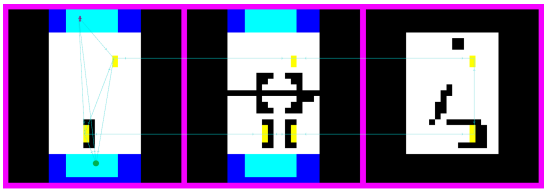

Fig. 8. Graph for path-finding between platforms. Edges of graph (Light blue lines) The portals, specifically stairs (yellow rectangles), passenger (purple person), and destination cell (green circle)

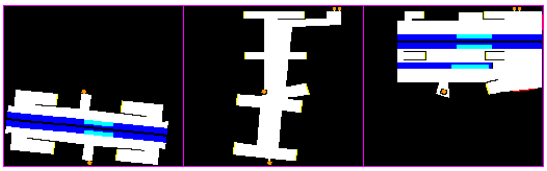

Fig. 9. Map for Trindade's station in NetLogo

- Number of ticks: The number of ticks that it takes to run the experiment and for every passenger to reach their destination I
- Distance per tick: The distance a passenger travels in a tick. By definition should be 0.3 units II
- Total distance: The total distance traveled by a passenger to reach their destination. III
- Crowdedness: The number of other passengers inside the given radius. IV, V, VI, VII

When analysing the number of ticks of each experiment, we can see that not always does the experiment of 300 passengers take more steps to run. Additionally, the scenarios with no elevators or a single set of stairs do increase the number of ticks, as expected, since passengers will not always be able to take what would otherwise be the shortest path. There doesn't seem to be a great difference between the base scenario and the scenario with escalators, which leads us to conclude that maybe the shortest path does not include the escalators. (or if it does, there is no significant benefit). One thing to notice is that there seems to have been some abnormal behavior in experiments *trindade-300* and *trindade-single-stairs* as is seen by the *max* column, which presents a much higher value. The percentiles are still within reason so we can interpret those values as outliers.

When it comes to the tick distance, the *max* column is where we should focus on. A tick distance higher than 0.3 happens when the collision mechanisms push a passenger out of the way. A 75th percentile of 0.3 is therefore expected as collisions are not that frequent. The higher number of passengers and scenarios with less pathing alternatives leading to an, even if slight, increase in the maximum amount is expected but inconclusive. To be certain, we'd need to have higher percentiles of the data to see if this relation holds not only for the tick with the maximum distance, but for the

ticks that employ some collision mechanism.

When it comes to totally distance travelled, mean values for each scenario are according to the hypothesis that less available pathing alternatives lead to worse outcomes. The scenarios *trindade*, *trindade-long-trains*, and *trindade-escalators* having the same paths available have a similar mean distance traveled, where as *trindade-no-elevators* has a higher distance traveled, and *trindade-single-stairs* higher even. Looking in particular at the scenario with single stairs, there is a 100 units difference between the experiment with 150 and 300 passengers. This distance could be explained by a worse starting position for a passenger. One thing to notice is that there is an almost perfect correlation between total distance and number of ticks. As such, referring back to the maximum value for the number of ticks in the *trindade-300* experiment which isn't reflected in the maximum distance traveled, we could hypothesize that that passenger might have become stuck, maybe as a malfunctioning of the simulation.

When it comes to crowdedness metrics, we can see that the numbers for the experiments with double the number of passengers are more or less double that of the normal amount. This is to be expected. The metrics for the scenario with the long trains is significantly lower, which makes sense considering the starting positions of the passengers will be more spaced out. There appears to be little correlation between the other scenarios and the crowdedness otherwise, which could indicate that the starting position of the passengers is the critical criteria for the crowdedness of the experiment. The results for a crowdedness of 0.6 are mostly 0, meaning the radius is too low to obtain any relevant information.

We can see how the differences in the scenarios did not always reflect in the data as would be expected from intuition. This could mean that the simulation is not a truthful interpretation of the real world or that the scenarios presented present insights into the dynamics of people in a transit station.

V. Conclusion

In this work, we used NetLogo to develop a simple pedestrian simulation of a transit station, with nothing more than an image as a guide for the modeling of said station. The implemented features would allow for the testing of most transit stations, given that common infrastructure was considered. The results provided some promising results but were not sufficient to validate the model. Repetition of the experiments, a deeper analysis of the data, and other metrics could be employed to increase the value of the simulation. In future work, passenger spawning could take a more realistic approach instead of passengers being all initialized at the start of the simulation. Additionally, path-finding algorithms that tackle the problems seen in Figure 3 could be fruitful. Different types of agents, for example, agents with incomplete knowledge or different path preferences, were disregarded in this work. Studying other stations and how they compare in terms of metrics could provide insights into their relative performance. Despite some of the advantages of how *turtles* and *patches* are handled, NetLogo may not be the most suited tool for more extensive work in this area. Nonetheless, the methodology and algorithms developed may still provide guidance on how to implement a different system from scratch.

References

[1] *Assessing Moscow Metro Station Blueprint with Crowd Simulation Software*. WEBSITE. URL: https://www.anylogic.com/resources/case-studies/assessing-moscow-metro-station-blueprint-with-crowd-simulation-approach/.

[2] *NetLogo Home Page*. WEBSITE. Accessed on 2024-01-10. URL: https://ccl.northwestern.edu/netlogo/.

[3] Marios Ntaflos. *Success Story — Buchanan helps Ferrocarrils de la Generalitat de Catalunya Transform Plaça de Catalunya Station into a Highly-Functioning Transport Hub*. WEBSITE. URL: https://blog.virtuosity.com/catalunya-station-gets-legion-pedestrian-simulation.

VI. Annexes

	mean	std	min	25%	50%	75%	max
trindade	506.55	184.89	133.00	430.75	537.00	626.00	873.00
trindade-300	515.65	190.14	141.00	415.50	556.00	617.00	1352.00
trindade-long-trains	492.73	199.75	144.00	299.25	533.00	622.50	902.00
trindade-long-trains-300	504.73	174.67	131.00	370.25	556.50	617.00	896.00
trindade-no-elevators	596.83	229.77	120.00	528.00	625.00	751.25	1040.00
trindade-no-elevators-300	626.60	225.73	126.00	531.00	645.50	763.25	1070.00
trindade-single-stairs	722.95	305.63	157.00	619.25	777.00	927.00	1393.00
trindade-single-stairs-300	604.53	228.16	137.00	521.00	631.00	750.25	1053.00
trindade-escalators	511.70	192.74	145.00	438.00	537.00	625.75	886.00
trindade-escalators-300	496.96	180.59	138.00	425.00	536.00	597.50	914.00

TABLE I

STATISTICS ON THE NUMBER OF TICKS FOR EACH OF THE EXPERIMENTS

	count	mean	std	min	25%	50%	75%	max
trindade	75983.00	0.30	0.02	0.00	0.30	0.30	0.30	0.55
trindade-300	154696.00	0.30	0.03	0.00	0.30	0.30	0.30	0.60
trindade-long-trains	73910.00	0.30	0.02	0.00	0.30	0.30	0.30	0.55
trindade-long-trains-300	151418.00	0.30	0.02	0.00	0.30	0.30	0.30	0.72
trindade-no-elevators	89524.00	0.30	0.01	0.04	0.30	0.30	0.30	0.58
trindade-no-elevators-300	187979.00	0.30	0.01	0.04	0.30	0.30	0.30	0.60
trindade-single-stairs	108443.00	0.30	0.01	0.00	0.30	0.30	0.30	0.56
trindade-single-stairs-300	181358.00	0.30	0.01	0.02	0.30	0.30	0.30	0.58
trindade-escalators	76755.00	0.31	0.09	0.00	0.30	0.30	0.30	0.90
trindade-escalators-300	149088.00	0.31	0.09	0.00	0.30	0.30	0.30	1.01

TABLE II

STATISTICS ON THE DISTANCE TRAVELED IN EACH TICK FOR EACH OF THE EXPERIMENTS

	mean	std	min	25%	50%	75%	max
trindade	150.91	55.35	39.45	126.78	160.35	186.80	260.37
trindade-300	152.61	54.36	42.14	122.74	164.21	184.42	261.64
trindade-long-trains	146.91	59.63	43.02	89.77	158.80	185.58	269.82
trindade-long-trains-300	150.30	52.24	39.11	109.93	166.42	183.99	267.25
trindade-no-elevators	179.03	68.91	36.18	158.66	187.01	225.32	311.86
trindade-no-elevators-300	187.86	67.68	37.40	159.77	193.72	228.52	320.38
trindade-single-stairs	216.79	91.81	47.02	185.98	232.98	278.26	418.16
trindade-single-stairs-300	181.24	68.40	40.84	156.70	189.19	224.80	316.42
trindade-escalators	158.36	60.68	43.15	129.36	173.33	193.04	264.95
trindade-escalators-300	154.30	57.44	41.47	126.23	170.84	190.38	273.52

TABLE III

STATISTICS ON THE TOTAL DISTANCE TRAVELED FOR EACH OF THE EXPERIMENTS

	mean	std	min	25%	50%	75%	max
trindade	0.00	0.07	0.00	0.00	0.00	0.00	4.00
trindade-300	0.02	0.21	0.00	0.00	0.00	0.00	6.00
trindade-long-trains	0.00	0.04	0.00	0.00	0.00	0.00	3.00
trindade-long-trains-300	0.01	0.20	0.00	0.00	0.00	0.00	6.00
trindade-no-elevators	0.00	0.02	0.00	0.00	0.00	0.00	1.00
trindade-no-elevators-300	0.01	0.10	0.00	0.00	0.00	0.00	3.00
trindade-single-stairs	0.00	0.05	0.00	0.00	0.00	0.00	3.00
trindade-single-stairs-300	0.01	0.09	0.00	0.00	0.00	0.00	4.00
trindade-escalators	0.00	0.08	0.00	0.00	0.00	0.00	4.00
trindade-escalators-300	0.02	0.20	0.00	0.00	0.00	0.00	7.00

TABLE IV

STATISTICS ON CROWDEDNESS AT A RADIUS OF 0.8 FOR EACH OF THE EXPERIMENTS

	mean	std	min	25%	50%	75%	max
trindade	0.28	0.63	0.00	0.00	0.00	0.00	5.00
trindade-300	0.66	1.01	0.00	0.00	0.00	1.00	8.00
trindade-long-trains	0.13	0.39	0.00	0.00	0.00	0.00	4.00
trindade-long-trains-300	0.40	0.78	0.00	0.00	0.00	1.00	9.00
trindade-no-elevators	0.22	0.53	0.00	0.00	0.00	0.00	4.00
trindade-no-elevators-300	0.67	0.96	0.00	0.00	0.00	1.00	5.00
trindade-single-stairs	0.26	0.59	0.00	0.00	0.00	0.00	5.00
trindade-single-stairs-300	0.68	0.93	0.00	0.00	0.00	1.00	6.00
trindade-escalators	0.30	0.67	0.00	0.00	0.00	0.00	6.00
trindade-escalators-300	0.57	0.98	0.00	0.00	0.00	1.00	9.00

TABLE V

STATISTICS ON CROWDEDNESS AT A RADIUS OF 1 FOR EACH OF THE EXPERIMENTS

	mean	std	min	25%	50%	75%	max
trindade	0.51	0.80	0.00	0.00	0.00	1.00	7.00
trindade-300	1.02	1.20	0.00	0.00	1.00	2.00	10.00
trindade-long-trains	0.27	0.55	0.00	0.00	0.00	0.00	5.00
trindade-long-trains-300	0.66	0.98	0.00	0.00	0.00	1.00	12.00
trindade-no-elevators	0.57	0.77	0.00	0.00	0.00	1.00	4.00
trindade-no-elevators-300	1.07	1.12	0.00	0.00	1.00	2.00	6.00
trindade-single-stairs	0.51	0.75	0.00	0.00	0.00	1.00	6.00
trindade-single-stairs-300	1.03	1.09	0.00	0.00	1.00	2.00	6.00
trindade-escalators	0.55	0.88	0.00	0.00	0.00	1.00	7.00
trindade-escalators-300	0.91	1.21	0.00	0.00	0.00	1.00	10.00

TABLE VI

STATISTICS ON CROWDEDNESS AT A RADIUS OF 1.2 FOR EACH OF THE EXPERIMENTS

	mean	std	min	25%	50%	75%	max
trindade	0.59	0.90	0.00	0.00	0.00	1.00	8.00
trindade-300	1.21	1.36	0.00	0.00	1.00	2.00	14.00
trindade-long-trains	0.34	0.63	0.00	0.00	0.00	1.00	6.00
trindade-long-trains-300	0.80	1.13	0.00	0.00	0.00	1.00	15.00
trindade-no-elevators	0.64	0.82	0.00	0.00	0.00	1.00	5.00
trindade-no-elevators-300	1.24	1.24	0.00	0.00	1.00	2.00	7.00
trindade-single-stairs	0.58	0.82	0.00	0.00	0.00	1.00	7.00
trindade-single-stairs-300	1.21	1.20	0.00	0.00	1.00	2.00	7.00
trindade-escalators	0.66	1.00	0.00	0.00	0.00	1.00	8.00
trindade-escalators-300	1.08	1.41	0.00	0.00	1.00	2.00	12.00

TABLE VII

STATISTICS ON CROWDEDNESS AT A RADIUS OF 1.4 FOR EACH OF THE EXPERIMENTS