Luna Park Simulation in Anylogic

Alice Schiavone

Project for the course "Simulation", held by Prof. Alberto Ceselli at Università degli Studi di Milano Statale

1 Introduction

Our Luna Park has two main sources of income: the entrance to the park is free, but each ride (like the Carousel or the Drop Tower) has a price, but a hungry visitor will also have to pay for snacks. If the visitor is feeling tired they can use a bench, or stop at a bathroom. At the end of the day (from 9:00 to 21:00, 12 hours) we are interested in knowing our profits, with respect to some other factors, like the number of visitors coming in (which could be influenced by advertising) or the number of facilities that satisfy visitors' needs. To simulate the day, we implemented a simple version of the Luna Park in AnyLogic (AnyLogic 8 Personal Learning Edition 8.7.12). The goal is to study the relationship between the Key System Parameters and the Key Performance Indicators: the first indicate the parameters that build and make the system work in a certain way, while the second show the performance of the system, which for us could be the money made that day.

2 Overview

Our LunaPark simulation will stop after 12 hours (or 720 minutes). In the meantime, visitors will enter the park and their main interest will be that of choosing a ride to go on. They can choose from the following, each with their parameters:

Name	X	Y	Price	Riding Time	Excitement	Intensity	MaxLineLength	Capacity
Carousel	200	200	€ 2,00	10	1	1	30	20
RollerCoaster	300	200	€ 8,00	2	10	10	100	10
Haunted House	400	200	€ 3,00	20	4	6	30	10
Swing Boat	200	300	€ 5,00	10	8	6	40	30
Drop Tower	300	300	€ 5,00	5	9	9	40	10
Log Flume	400	300	€ 6,00	15	7	2	40	20
River Rapids	200	400	€ 4,00	15	6	4	40	20
Bumper Cars	300	400	€ 3,00	10	5	2	30	20
Ferris Wheel	400	400	€ 2,00	20	3	3	70	50

X, Y	Location on the main frame.
Price	Price of the ride for one visitor.
Riding Time	Time duration for riding the attraction.
Excitement	How exciting it is to go on this ride.
Intensity	How intense it is to go on this ride.
MaxLineLength	Maximum queue length for the ride.
Capacity	Maximum number of people allowed together on the ride.

The parameters describe:

After going on a ride, each visitor will check their needs. If some need has to be satisfied, the visitor will look for a Bench (if tired), a Stall (if hungry) or a Bathroom. If any of these needs cannot be satisfied, the visitor will go home (with their unspent money!). We want our visitors to stay as long as possible doing rides or buying food. The visitor logic is described by its flowchart:

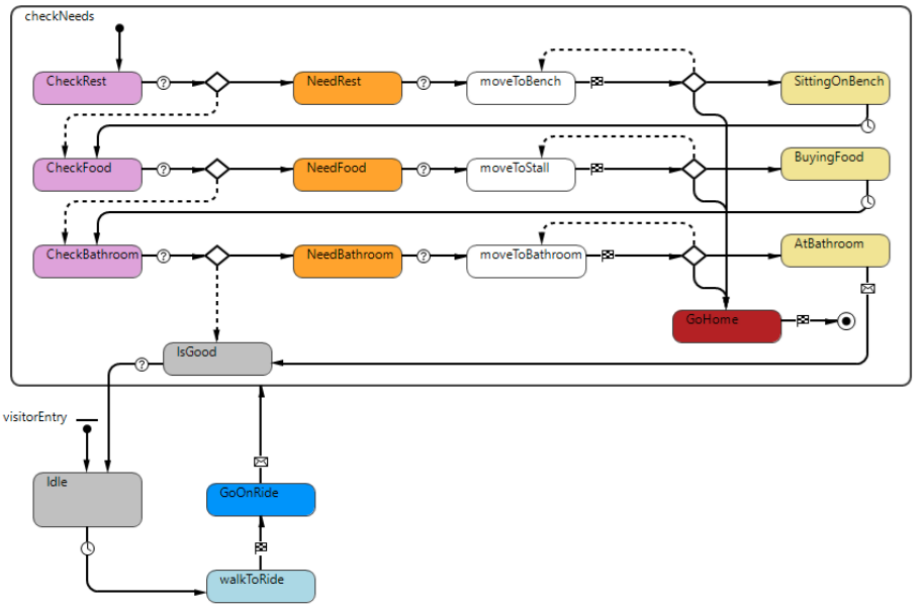


Figure 1: Blabla

To avoid cluttering this report, details about the logic of the Visitor agents, like parameters and thresholds, can be inspected by looking at the Anylogic model code. It is worth mentioning that the patience and the money parameters are decided at agent creation, following (respectively) a truncated normal and an exponential distribution.

Money	exponential(0.05, 10)	[lambda, min]
Patience	(int) normal(30, 120, 60, 20)	[min, max, shift = mean, stretch = std deviation]

The facility or ride is selected uniformly at random.

The visitor needs to have their needs above a certain threshold (20) for that need to be satisfied. Needs are:

1. Energy
2. Hunger
3. Bathroom Need

When any of these are less than 20, the visitor will try to raise its need points. If they can't, they will leave the park. At agent generation each need has points equal to 100. The function LowerNeeds decreases point based on elapsed time. If the visitor that is trying to satisfy their needs and meets a full line, or doesn't have enough money, will try another facility by lowering its needs further. If the need points reach zero or lower, the visitor will go home.

The visitor Excitement and Intensity are not actually useful to the analysis or to the simulation, but they still give insights about the level of satisfaction of a visitor, and could be considered secondary KPIs.

The logic of the Visitor relies on 4 other agents, briefly described as follows.

2.1 Ride

When selecting a ride, a visitor will firstly check if the line is full, if not they will join the queue. Visitors in line will go on the ride if the capacity of the ride is not reached. The ride will start only when the capacity limit is met, or otherwise after some time is passed. The events WaitForResume and ResumeIfFull handle this process. The visitors are put on hold if the capacity is met, and the delay (riding time) is started by the two events. If a visitor stays in line for too long (exceeding the patience parameter), it will exit the ride queue.

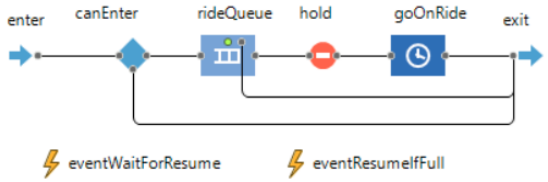


Figure 2: Rides state chart.

2.2 Bench

Benches have a capacity of one visitor, and will light up in red when in use. Visitors will sit on a bench for 10 minutes and restore 50 energy points.

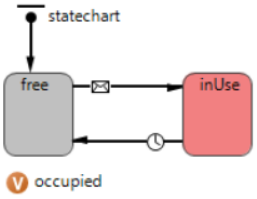


Figure 3: Bench state chart.

2.3 Stall

Stall are generated based on a table with attributes “Name” and “Price”. The hunger value restored by each stall is based on the price, so a pricier food will restore more hunger points. We assume that a visitor is served as soon as he orders food at a stall.

2.4 Bathroom

Bathrooms follow a simple queue-delay logic, with a capacity of 10 bathroom stalls per bathroom.

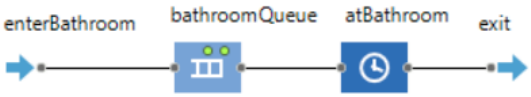


Figure 4: Bathroom state chart.

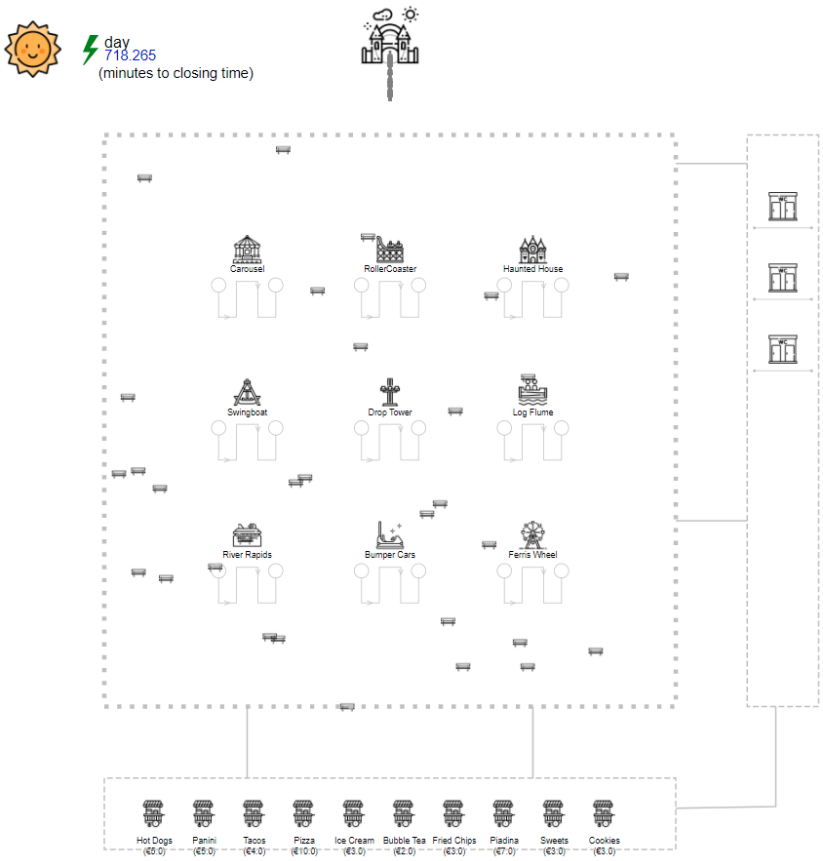


Figure 5: The LunaPark at the beginning of the day.

3 Key System Parameters and Key Performance Indicators

We have many parameters for this system, but we will call KSPs (Key System Parameters) only those parameters that we can imagine to be actually modified in a real life scenario. For example, the threshold for which a visitor will look for a bench is not a KSP, because us as Luna Park owners cannot have an impact on that. Instead, we will focus particularly on the following:

- Total Profit
  Total amount of money spent in the park.
- Profit per visitor
  Average of total profits by number of visitors.

- Unspent money (Total and mean per visitor)
  Money that visitors had when arriving in the park, minus the money spent.

- Impatience Visitors
  Number of visitors that will leave the ride queue because they have waited too long.

Other parameters like the price of the rides or the food influence the system, but we will treat them as immutable and agnostic to the system. We will not actually define a price for our advertising strategies, because that is behind the scope of this project. Instead, we will ask ourselves: "If I can reach a certain number of visitors per minute, how much profit can I make?".

To evaluate the performance of the model, we need to identify on which specific index we want to focus our attention. For us, the KPIs (Key Performance Indicators), will be:

Money	exponential(0.05, 10)	[lambda, min]
Patience	(int) normal(30, 120, 60, 20)	[min, max, shift = mean, stretch = std deviation]

For the runs on multiple variables, we will focus on the relation between profits (total and per visitor) with respect to the number of visitors per minute, and the number of benches and bathrooms in the park.

4 Experimental set up

Our simulation will run for 12 hours (720 minutes) and it will include 9 attractions, 10 food stalls and a variable number of bathrooms and benches. The most important parameter is visitors per minute. They will enter from the main gate and walk across the main park area with attractions and benches. For the food stall we have a separated food district, and the same holds for the bathrooms. The model is run with variable parameters using the Experiment function of AnyLogic, with the following values:

Parameter	Min	Max	Step
visitorsPerMinute	1	3	5
nBenches	20	50	10
nBathrooms	2	4	1

Because of the stochastic nature of the model, each iteration will run for 10 times to ensure a more reliable statistical result.

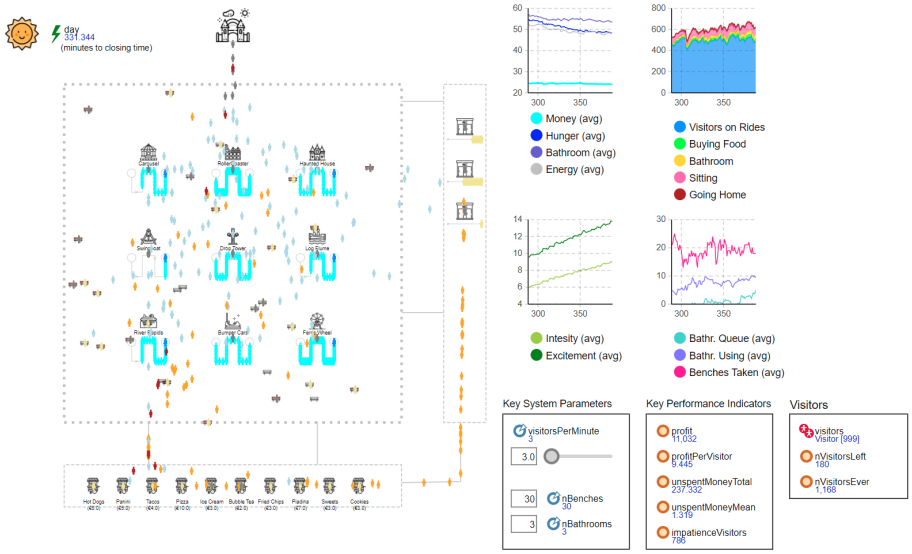
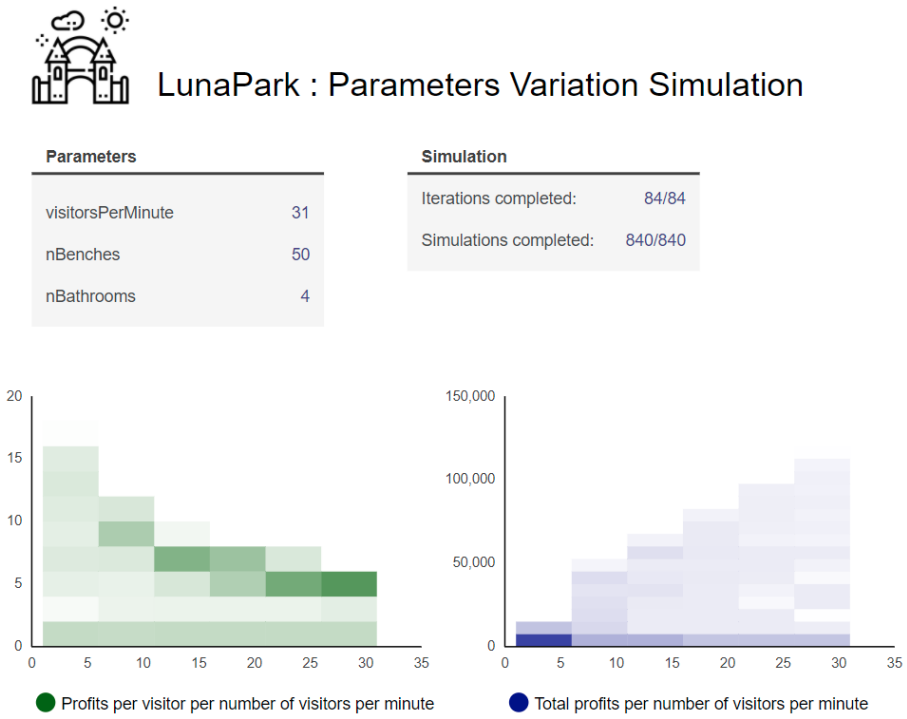


Figure 6: The LunaPark the during the day.

5 Results

We can build a chart with the number of visitors per minute on the X-axis, and the total profits (■) or profits per visitor (■) on the Y-axis. This allows us to see the relation between the two variables, but because we are running multiple iterations with different parameters (number of benches and bathrooms), we can see how these affect the final results.

If we look at (■), we can see that for a very low number of visitors (1-5), having a different number of benches or bathrooms does not affect the final outcome in total profits. As we increase the number of visitors, we see that our model is able to reach a profit of over 100,000€ with at least one configuration, and in general the variability in results suggests that a different number of benches or bathrooms affects total profits when more visitors are coming to the park. However, if our goal is to maximize the profit per visitor, so the money spent by each visitor before going home, by looking at (■) we can see that the relationship is reversed: more people coming means less opportunity for the other visitors to spend money, converging to a very low value (4€-6€) per visitor even with different bench-bathroom configurations.



6 Conclusions

A LunaPark model is, like any other process that involves a crowd and business venue, a complex process to model. Nonetheless, by running a simulation that approximates the real life situation, we can estimate how much money can be made, or see how a crowd could behave in it. Due to the stochastic nature of the process, given by the use of random variables, more statistically reliable results are achievable with different levels of confidence by running the model multiple times. We have seen that the number of facilities affect how much money people will spend before going home, and that the number of visitors per a period of time (which could be influenced by a marketing campaign) is not the only variable in the equation. It would be interesting to also measure how much time visitors spend in the park, and if this has a relation with the money or fun (modeled by the excitement-intensity variables). This last concept could be expanded and researched more, for example by increasing the number of people coming if previous visitors actually had fun (or vice versa).

Acknowledgements

This project has been developed entirely by me and supervised by Prof. Alberto Ceselli. For more information about AnyLogic and its components, refer to the AnyLogic documentation at anylogic.help. Icon credits to Flaticon.