# Simulation Masterclass

## Assigment 2 - group 14


| | |
|---|---|
| Claudio Prosperini | 6084931 |
| Juliëtte van Alst | 5402409 |
| Gerben Bultema | 5309247 |
| Roelof Kooijman | 5389437 |


TU Delft

SEN9110

# Content

- Practical information
- Description animation
- Animation features
- Monitors
- What was easy and what was difficult
- Comparison Salabim, MESA and SD

# Practical information

- Link to Github repository: https://github.com/roesel1/SEN9110-G14,
- The correct version has the commit message: FINAL ASSIGNMENT 2 -with video and slides
- The basic supermarket model is in the file *ASSIGNMENT_2_supermarket_layout_animations_model.ipynb*
- *A requirement* file is added. If it does not work then creating a env with python 3.10 and pip installing salabim,pandas,numpy, notebook, jupyterlab, seaborn and matplotlib should work
- Two videos are in the Assignment 2 folder from which slow video shows the opening of the supermarket with its first customer. The long video is a sped up version for the whole supermarkted

# Description of how animations were added

The animation's layout was based on the image in assignment 1. To get the proper dimensions, draw.io's rulers were used. This can be seen in the code as locations are given by the translation to center plus the ruler coordinate *40. This gives a scale of **5 cm per pixel**. The next slide shows the comparison of the image of assignment 1 and the representation in Salabim.

**2D animation** was chosen as running the example 3D models gave errors. This saved time and didn't change the installation requirements.

Customers are created using ellipses. Each time they start shopping for a new product they get a new **trajectory** that they walk. The starting point is based on their current location resulting in smooth movement. Movement is not physics based and is separated from the model, so it is possible that if a customer is especially quick in getting their items that they theoretically could be in a different section than they are visually in and walk through shelves to get there.

Monitors are displayed below the store's animation indicating the amount of customers present and time spent in the store.

Customers, queues and monitors were all displayed using the built in animation classes. This made it very easy to display them.

The order of creation of the various components for the visualization in the code is given the flowchart with yellow coloured nodes (given in on of the next slides). As depicted in the flowchart for initialisation the creation of the visual components is done at the end of the model initialisation.

# Description of how animations were added

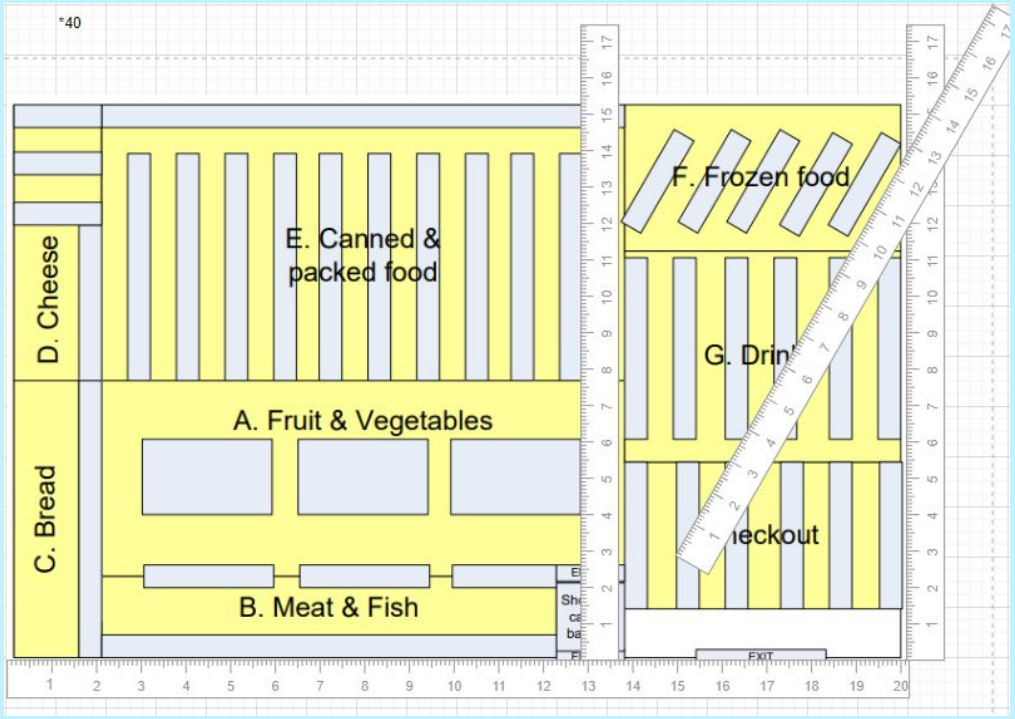



Image 1: how dimensions were derived

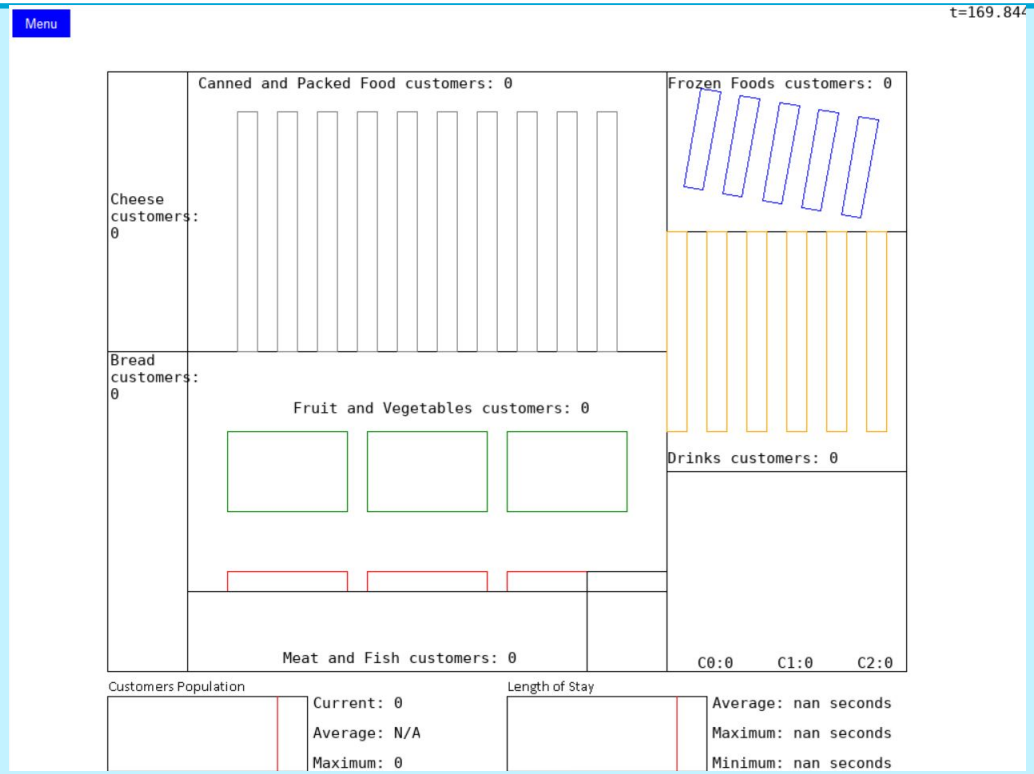
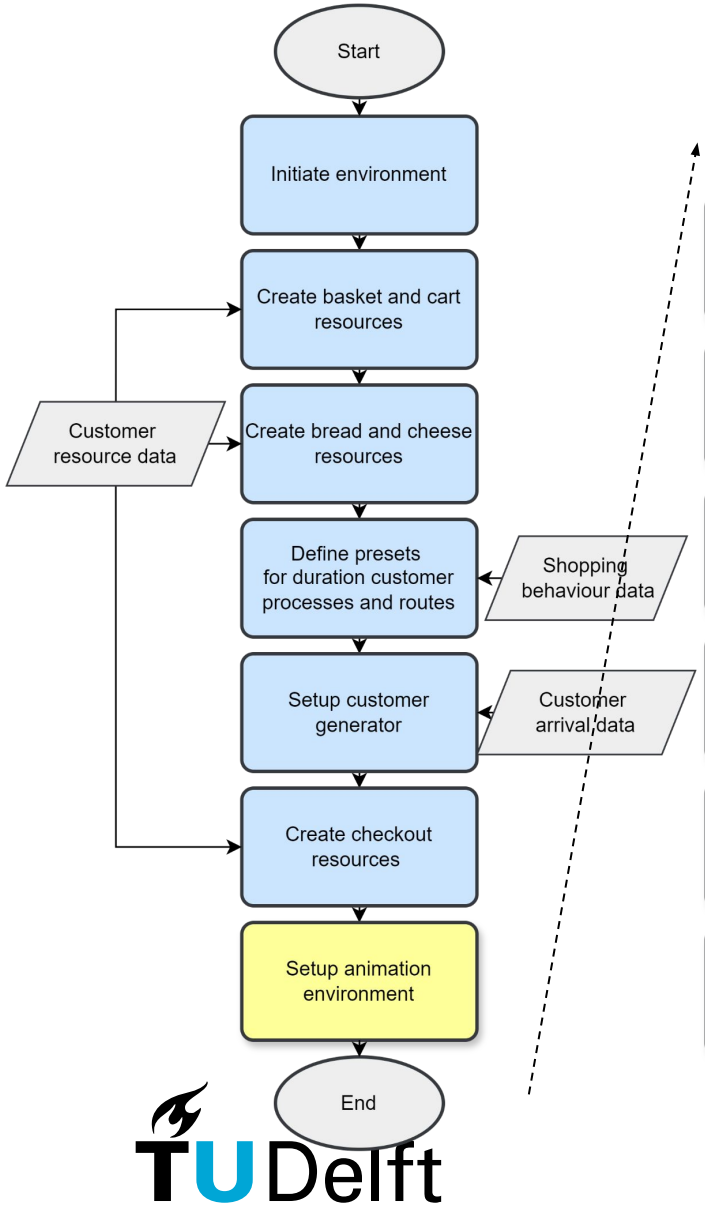
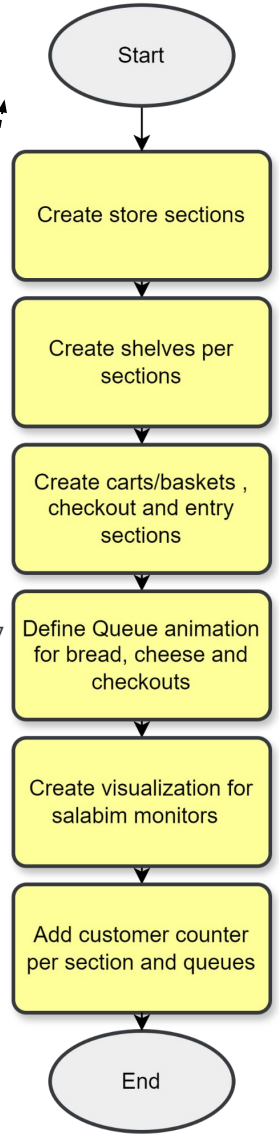
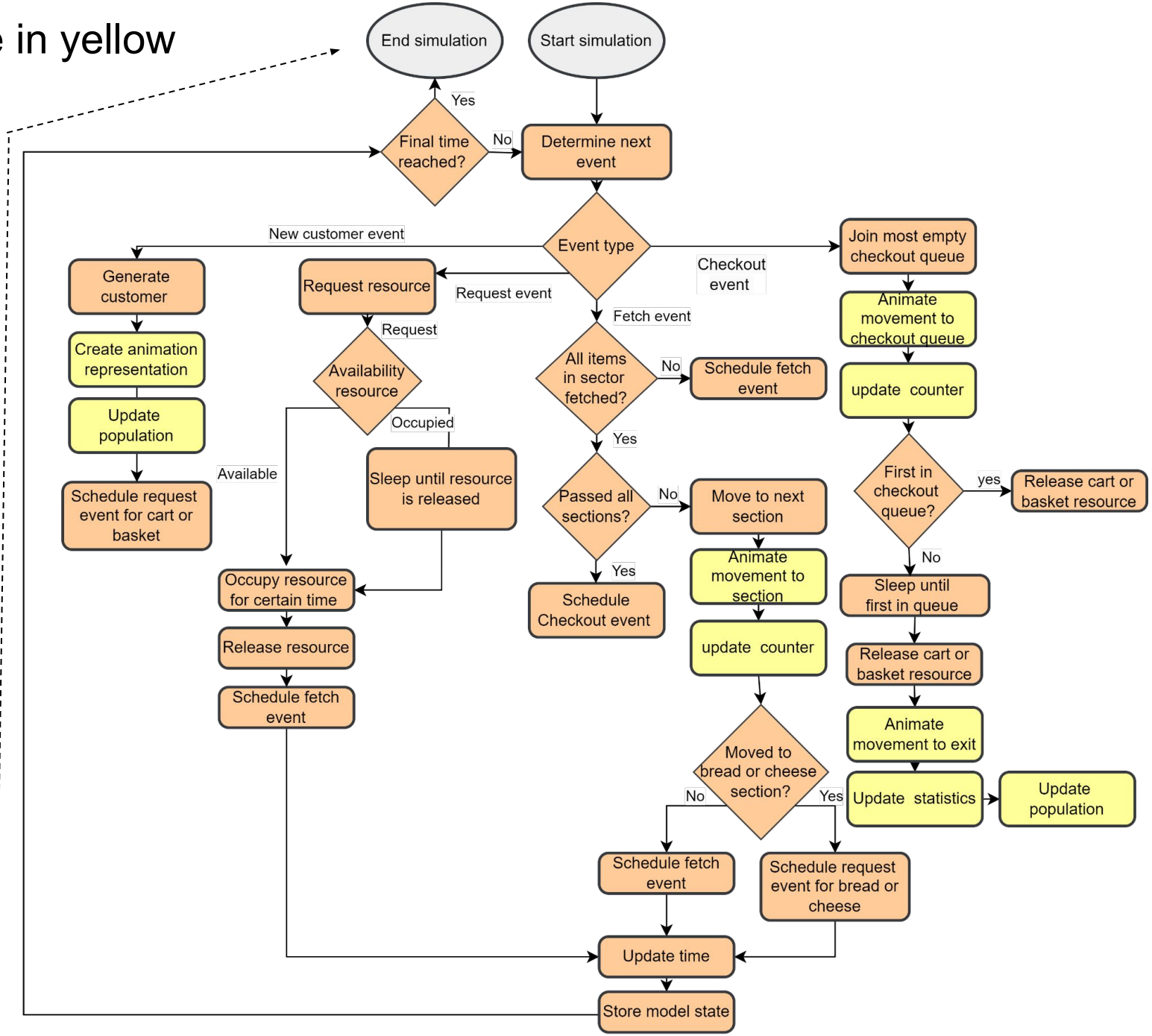



Image 2: Representation Salabim

# Animation components added are in yellow




TU Delft

# Animation Features - Text and Monitors

To animate live data over the run of the simulation, we used Salabim's .AnimateText() and .AnimateMonitor() functions.

Both these functions work well with native Salabim objects like queues and resources, making it easy to track and display certain statistics. The positioning of the Text and Monitor boxes is intuitive. The scaling options, particularly the vertical one, is essential to represent the data in a meaningful way: without proper adjustment, most data points (especially with Length of Stay) end outside the upper bounds of the Monitor boxes, making them uninterpretable.

While the need for a queue/resource item creates a good synergy between animation and native Salabim components, it can also be a demerit. In order to represent the information required by the assignment, artificial queues had to be initialized, both for each department and the whole supermarket. This did not complicate the simulation in a meaningful way, but it would have been more functional to create specific monitors with updating tallies. For instance, a supermarket_population_monitor.tally(+1) was initially conceptualized as a customer monitor updater within every customer's process (right after start_shopping), but this feature did not produce the expected outcome and was discarded.

# Monitors - Level vs Non Level

Defining custom monitors would have been especially helpful with determining whether a monitor is to be level or not. The “default” monitors of queue items, which in the specific use case were always .length(), .length_of_stay(), or some variation of the two, are respectively level and non level.

Level monitors can natively display and update their current value based on the state in the simulation, which makes them very apt for representing the current population of customers in the supermarket at all times, as well as other interesting statistics such as .mean() or .max().

With the case of .mean() in particular, it should also be noted that there is a feature to exclude 0 values from the calculation, allowing for models with a “start-up” phase like this one to not have polluted data. When using this feature with something like round(), however, the NaN value of the .mean() function during the start-up time needs to be properly handled.

Non level monitors, on the other hand, do not support the .get() function that allows to display and update the current value of a variable based on the last observation. As this value would not be on the animation screen for long, this feature was not implemented, in favor of displaying the minimum throughput time. If, however, there was an interest in creating a scatter plot of the throughput time data against each customer, a non-native workaround would need to be devised.

# What was easy and what was hard

1. Adding angled shelves in the frozen foods section was surprisingly difficult as salabim, for some reason, applies the translation (x and y coordinates) before applying the rotation. Thus the rotation is not relative to the animated object, but to the entire screen. This made it significantly harder to properly position the shelves.
2. Additionally we ran into issues with the window size. We wanted to add extra space for the monitors but increasing window size scales the entire animation is rather unpredictable ways. We were not able to figure out how to disable this and thus went with the smaller screen.
3. Furthermore in contrast to some of the packages that were displayed during the monday lecture, creating the layouts and paths was rather tedious as all rectangles had to be created separately from coordinates. This would be much easier with dragging and dropping into the environment.
4. Documentation provides examples but rarely precisely tells how things work. E.g. it is stated that the shape of the queue objects can be changed. However, the way to do it is not explained. This ended up being changeable in the component class, not the queue itself.
   Additionally some functionality is not included in the documentation e.g. how to remove objects from the simulation. The lacking documentation made developing the animations take much longer than anticipated.
5. We chose to keep the animation completely separate from the model and not make changes to the workings of the model itself. By separating the animation from the model run itself, the animation can be too slow and thus some snapping can occur when joining queues. Additionally if the animation is too slow customers will walk to the next section already, even through shelves. You could either speed up customers either in speed or by using timed animations or wait till animations are finished. In the first case the animation would have weird speeds, in the second the model is influenced.
6. While there is a Animation() class that allows for animations to take a specific duration, the trajectory class, which makes defining paths much easier, only allows for entering the speed and acceleration, not the time. In the end we chose to use this approach as this would lead to customers having a default, controllable speed, over customers randomly speeding up in certain sections to meet their ‘deadline’.
7. We ran into a bug where sometimes the customers seem to be invisible while in the checkout queue, as if they had already finished shopping or were added multiple times, however we could not figure out why this happens.
8. Animation was, after some struggles with documentation, easier to implement than it would have been in mesa. In mesa you are limited to the type of space you are working with.
9. Salabim also comes with some tools to debug the animations (e.g. you can display the polygons of polygon trajectories), which was very nice

# Comparison Salabim, SD and MESA

Visualization with SD would be hard because no information of individual customers are available. So, within Vensim the best visualization is the overview of stocks and flow, but which does not show an representation of the supermarkt nor does it show progression over time.

Compared to MESA salabim is a bit harder to start creating visualization but gives more freedom for adding specific elements. Assuming that one would make the supermarkt on a 2D grid then the model class requires an filled in grid with cells representing queues, shelves and checkouts. So visualization the setup is easier in MESA. However for routing visualisation Salabim is easier because it provides tools to calculate a trajectory using coordinates. MESA does not contain tools (so far as we know) tools so you have to make a function/method which defines the route and a function/method to visualise this. So MESA is easier to show the environment but harder to quickly define movement as an visualization

Addiotionally, when using a grid, space will be defined by discrete distances which does not allow for realistic waiting behavior before the checkout because nobody stance in perfect square like distance of each other. This could be solved by creating a very fine grid, but this is not optimal.

*Table 1: Differences between the simulation package*

| Aspect | SD using Vensim | DES using Salabim | ABM using MESA |
|---|---|---|---|
| Representation | Non-spatial representation, uses stocks per section | Inbuild visualisation tools focused on continuous space movement | Inbuilt functions to define space which can be visualised using MESA functions or extra libraries. Allows for networks, grids and continuous space. |
| Effort | Vensim already shows stocks and flow providing visualisation | Creating environment is not intuitive | Space defined for model self can be used |


TU Delft