Simulation Masterclass

Assignment 3 - group 14

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

TU Delft

SEN9110

Content

- Practical information
  - Checkout behaviour
  - Improving existing animations
  - Improving customer behaviour with collisions
  - Further improvements which we did not implement
  - What was difficult/easy (compared to other packages)

Practical information

- Link to Github repository: https://github.com/roesel1/SEN9110-G14,
- The correct version has the commit message: FINAL ASSIGNMENT 3
- The basic supermarket model is in the file *ASSIGNMENT_3_supermarket.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

Checkout behaviour

Customers choosing the checkout queue based on the items in front of them was really easy to implement. One line of code that determines how queues are chosen was changed:

```
emptiest_queue = min(checkouts, key=lambda checkout: checkout.requesters().length()*50+sum(i.item_count*1.1 for i
 in checkout.requesters()))
```

The lambda function determines the average time it would take the people in each line to be processed (given by payment time + item scanning time) and decides based on that.

Implementing clerk behaviour also was relatively problem free. In the customer component we defined the wait time. This made it so there is no synchronisation with activating and passivating needed between the customer and the clerk. We did add a new clerk component with two possible processes, but only to handle the animations. The customer would tell the clerk to perform one of two animations with a certain duration. Defining the clerk as a separate component made timing the animations easy as we could make use of the hold() functionality.

Improving the existing animation

In the last assignment a bug caused our animations to play too quickly. We fixed this and as a result were able to define more appropriate paths using the setup of the last assignment.

Additionally the animations were separated from the running animation, in which certain actions would sync the animation by either letting them wait at the end of their trajectory or snap to the proper location. In our new system we synchronised the trajectories with the taking of items. This was essential for creating the collision detection for carts later on. We did this by determining the exact times the customer should move and stop. To do this we divided the trajectory for the entire department and its duration t1() into equal parts for each of the items it will get in that department. As this time became rather long in some cases, we could not stick to the 20-30 seconds item taking distribution set in the first assignment. Instead we decided to use a new distribution for taking items from the shelves and have the walking speed determine the time it takes to get to the destination. We thought this would make sense as the time it would take someone to finish shopping with a lower speed (a cart) should logically be slower than those with a higher speed (carrying a basket).

.

Improving the customer behaviour: collisions

For each department with collision detection, the last customer with a cart that entered it is stored in a dictionary. This allows a new customer to know which customer it is following. This customer will ‘tell’ the customer in front it is following it in order to subscribe to its messaging stating it will be standing still for a certain amount of time. If a customer gets this message, it will determine how long it takes to get to the position of the customer that is waiting. It then will either continue its path, or walk up to the customer in front and wait for the amount of time. An alternative approach we considered that uses Salabim’s features was to use a queue to remember the order of cart-bearing customers in the department, however the aforementioned approach is more efficient.

Dealing with different customer speeds.
In order to deal with different customer speeds, we let customers check if the person in front is quicker or slower. If the customer in front is quicker, then there are no problems and the model continues as described previously. If the speed of the customer in front is slower, then the customer will determine the time it will take to arrive at the current location of the customer in front. After this time, the customer will check again. If it is within a certain threshold time (0.5s), it will match the speed of the customer in front. This temporary speed is reset once the customer stands still.

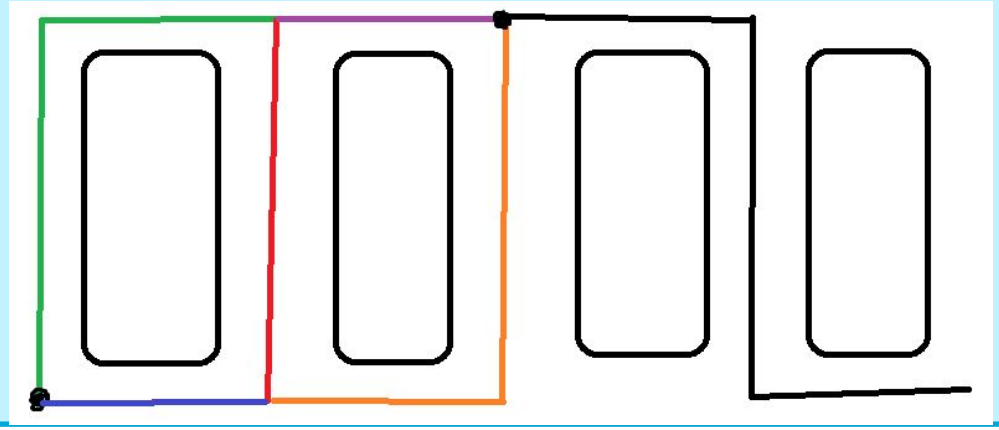
This all takes place while the customer is in ‘hold’. To make sure the model is delayed the correct amount of time, the extra time that is added by customers waiting for the customer in front and delayed by the lower speed is remembered and used in a subsequent hold statement. This also allows a monitor to gather data about how much customers are blocking each other.

Further improvements which we did not implement

We also thought of a few ways in which to make the model and animations more accurate. Our model was made in a way to make it extensible in the ways we envisioned. We decided against implementing these due to time constraints, but did want to showcase our ideas.

One limitation of our model is that customers are not able to make smart route choices in order to overtake slow customers in front of them. Customers could decide to walk different routes through a certain department. This is implemented in a way that is very similar to how routing for the entire department currently works. You would define multiple paths through the shelves that each start and end at the same place. Each of these would have its own queue and trajectory, similar to the behaviour described in the previous slide. At an intersection the customers can choose between the different routes e.g. based on the speed of the last cart in each route or the crowdedness. We did not implement this behaviour and pathing in the model as a conflict resolution algorithm would need to be implemented for when multiple customers arrive at the same intersection and we did not have time for this. An example of which is letting the closer customer pass, if they are at the same distance, letting the one with the higher speed pass and if that does not work choosing randomly.
An example of this is given in the figure below, where each coloured segment would have its own queue or last cart entered dictionary entry. The black line represents our current approach.

You could go even further and define two sides for each aisle you can walk on, in which you can walk around each other and set item locations at different places. However this is not relevant for the current assignment and would take a lot more time.



What was difficult/easy (compared to other packages)

We once again ran into limitations with Trajectories. Trajectories were not made to be interrupted halfway through in the way we use them. We hoped that by setting t0 it would resume the trajectory from the point it had left off, however this is not how trajectories work. This made keeping track of the position of customers relative to each other and restarting trajectories much more difficult. We added extra helper functions, that found points of the polygon already traversed, to make this slightly easier, but we believe there should be easier ways to achieve this by default in Salabim.

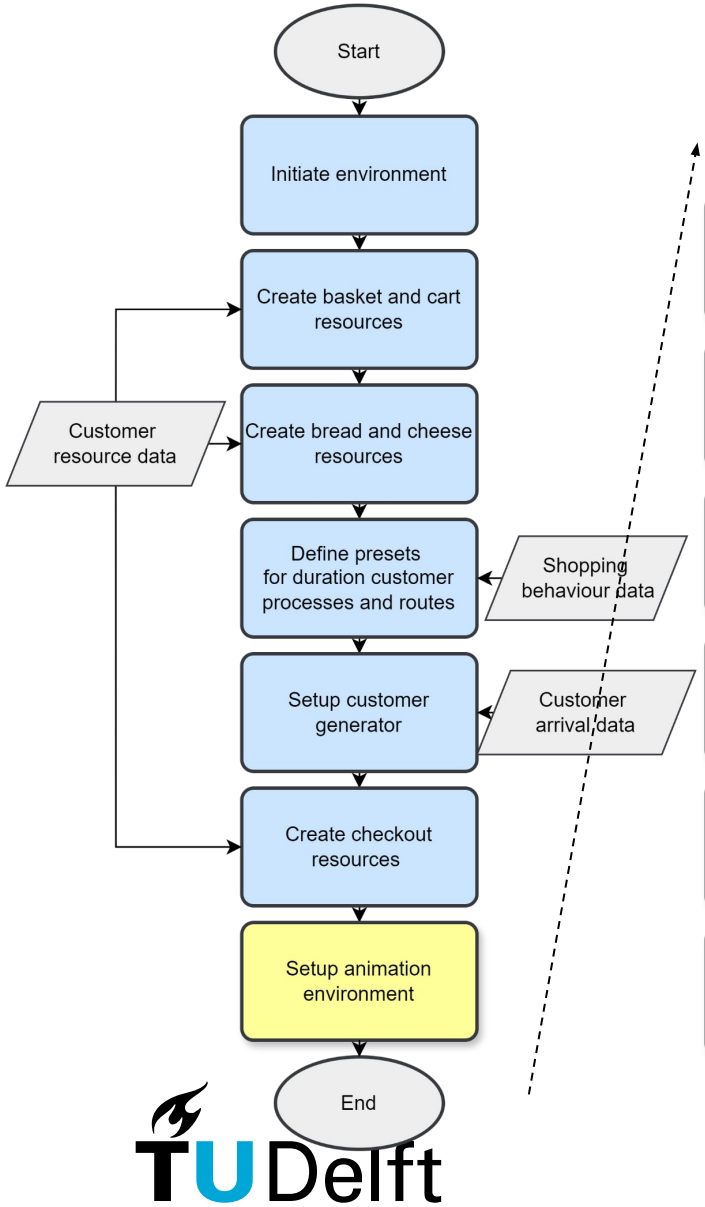
Additionally the customers now interacting with each other resulted in some weird behaviour, where one customer could tell to the customer behind it that it would stop, but then the customer behind it did know it had a customer ahead of it.

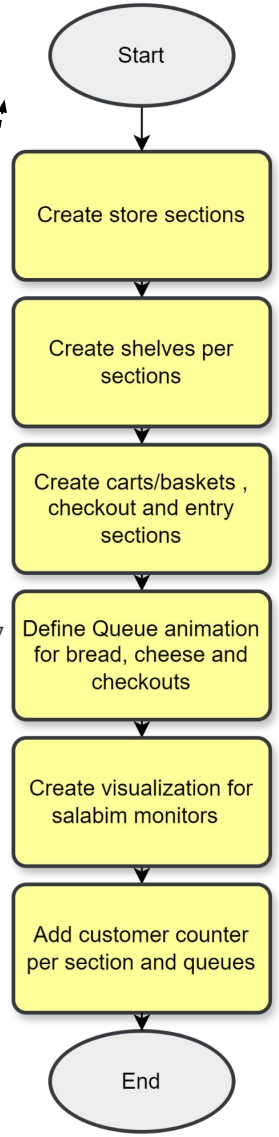
Despite the complications, it was still relatively easy compared to known packages such as mesa to get the collision detection and speeds working in a way that allowed for floating point numbers, at least conceptually and with the options to expand the model with smoother decelerations. In mesa we would likely have used a fine grid or continuous space and checked the cells in front on the path. It would also result in trouble with the order in which agents execute their movement as they all tick at the same instant. Especially since you would want to have the furthest along in the process go first in order for them to not accidentally block each other. This would result in a lot of ordering issues after the bread and dairy aisles.

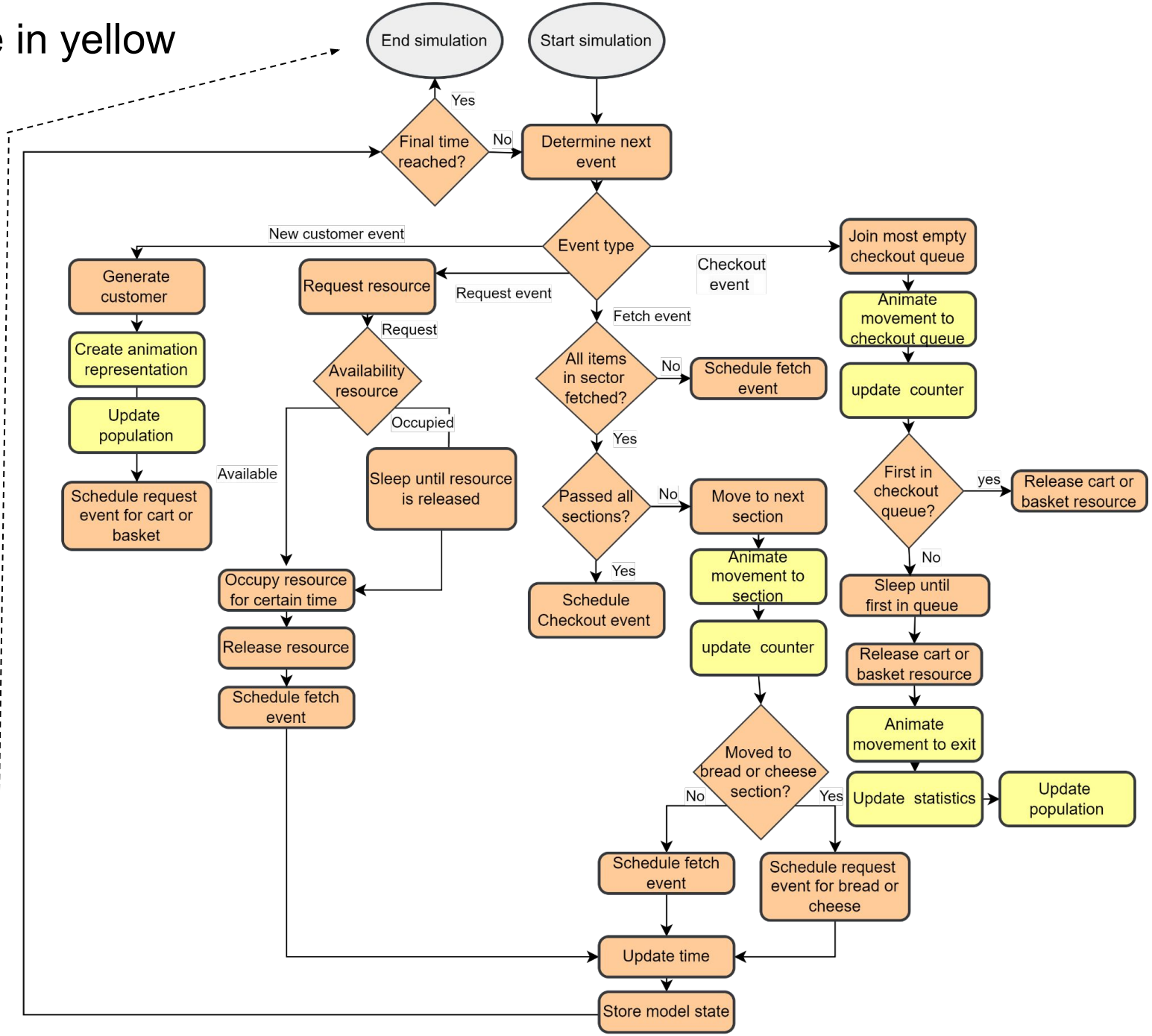
For the clerk checkout behaviour, the implementation would be very similar between the packages. The only difference is that Salabim includes a much easier to use/more advanced animation suite.


TU Delft

Animation components added are in yellow







TU Delft