Modeling and Simulation

Cooking Process Simulation

Martina Mlezivová, 12306360 - 033 266*
Lana Ramšak, 12302817 - 033 201†
Klaus Derks, 01529340 - 066 926‡
Dávid Lukács, 12306344 - 033 535§

February 10, 2024
Supervisor: Martin Bicher

Abstract

Doing tasks in the right order can be valuable to any kind of process. By finding the optimal order one can save time and resources. We were instructed to explore this, particularly in the context of cooking. In our Cooking Process Simulation project, we developed a Python model that, when given a recipe, identifies the most efficient task order. The order is dependand on the number of cooks and available resources.

We tested the model to understand when task order matters and whether the quantity of cooks and resources affects the overall duration. Surprisingly, having more than two cooks rarely impacted duration, while a different task proved to make a big difference. For cases with only one cook and tasks not dependand on the cook's presence, the order showed minimal importance.

*Did the results, introduction and discussion.
†Added testing permutations and optimized the code, for getting the best order,wrote the implementation and worked on results for the documentation.
‡Wrote the simulation code of the recipe
§Wrote the json files, documentation

Contents

1 Introduction

In the past years the industry put an emphasise on being the fastest and most efficient. Take-away is getting more and more popular. Next day delivery is not as rare as it used to be and products are rarely hand made. Optimizing the process plays a big role in that. A simple example can be cooking, where the order of tasks significantly impacts the efficiency. When following a recipe, you have to think in advance what you will need, and how much time it will take. If you don't turn on the oven before starting, you might waste time waiting for it to heat up later. In our project we will be trying to find an optimal ordering of tasks, to make the cooking process more efficient. This way recipes can be done quicker and the work can be better distributed.

For our starter recipe simulation we chose something basic. The recipe that came to mind was the dish goulash. Here is the following recipe written in steps:

- cutting the onions
- frying the onions
- cutting the meat
- cooking the meat
- peeling potatoes
- cooking potatoes in the previously boiled water
- cooking everything together

As you can see it includes a lot of different ingredients and cutting tasks, which are fairly simple to include in our simulation but help set the start model. Furthermore, it is a recipe where the order of the tasks is important, which is something you have to be aware of when cooking.

Throughout the project we also implemented two other recipes, potato salad and cake. One was highly based on a cooking book and was used to test wheather our order is equal to the one in the book. Lastly, we added the recipe for the cake, to better observe the influence of different numbers of cooks.

2 Model Implementation

Our program is written in the programming language Python. We made that decision because that is the program we are all familiar with and will all be able to understand the code.

2.1 Data storage

Tasks are implemented as objects, where each one has it is own

- `name`, stating the name of the task,
- `duration`, meaning the time it takes for the task,
- `required task-prerequisites`, which is a list of tasks-names that have to be completed before,
- `required resources`, meaning all items needed for the task for instance *cooking stove, knife, cook*

Our recipes are portrayed as lists of tasks and saved in a separate `json file`. The kitchen resources are also defined in another `json file`, where each one is assigned a `name` and `num`, meaning quantity.
A few parameters are set at the start of the code. These are the recipe file and the difference of time factor, which we will talk more about later. To save the data, that is changing throughout the cooking process, we constricted a class cooking, where we define these properties:

- list of completed tasks, which is at first empty, later on we will add the completed tasks here,
- list of resources, imported from the json file,
- list of tasks, also imported from the json file,
- copy of list of tasks, that will not change throughout the simulation,
- order, where we save the order of tasks in current simulation.

2.2 Recipe Simulation

For the actual simulation, meaning for a recipe to be excecuted, we defined another class named DiscreteEventSimulator. Everytime we construct a new one, we also construct a new class cooking. This way all our parameters are everytime set back to the start (ex. list of completed tasks). The class also has a parameter `event_queue`, where scheduled events are being added, and parameter `time`.

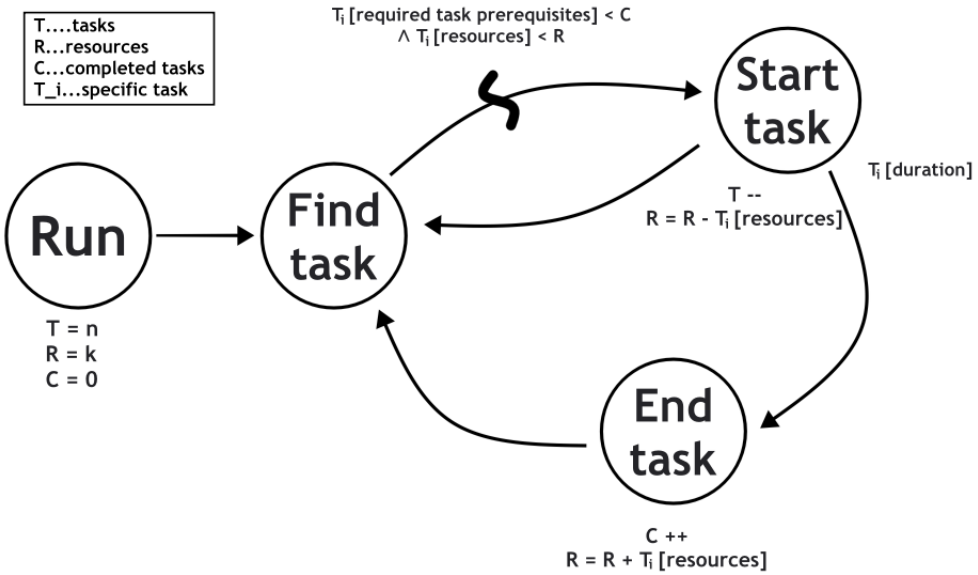

Figure 1: Discrete simulation graph of a cooking process.

On this figure 1, we can see a simplified graph on how the simulation works. We start by calling the method `find_task`, this as suggested searches for a task that can be executed, meaning all resources needed are available and all tasks, that needed to be done before, are already completed. If we find such a task, method `start_task` is called, where the previously mentioned resources are taken of the list, the task name is added to the order and the method `schedule_end_task` is invoked. There we make an event, where we calculate the time when it will be finished by adding up duration to 'current time' and save the name of the task. This is appended to the `event_queue` list. Directly after, the method `find_task` is executed again.

Right at the start we start a while loop, that will be done once. The number of completed tasks is the same as the number of tasks at the start. With each iteration of the loop, the time increases by one. We also use this to check if there is an event scheduled at the current time. In case there is one we call the method `end_task`. This is where the previously removed resources are added back to the list and the task is added to the list of completed tasks.

At the end the function `run_simulation` returns a tuple, where the first component is the time it took, and the second the order in which the tasks were completed. In case the you one does not have the required resources needed for the recipe, the program returns the error: `You do not have the right resources`.

Now that we got our program to execute a recipe, we started to work on finding the optimal ordering of tasks, to get the shortest time.

2.3 Different orders of tasks

The first idea we got, was just to try all possible permutations of the list of tasks. Run the simulation for each one of them and then choose the one with the shortest time. To help with understanding the code and making sure it worked well, we made sure to get a list of all possible orders and the time it took. Furthermore due to prior task prerequisites the recipe we had at the start returned us the same duration, no mater the order of tasks. To fix it we added a task, to have a three-task process, meaning heating the water, peeling potatoes and cooking them.

Moreover, we changed some of the durations of tasks.

When running the code we realized, that the order of the last few tasks rarely changes. This was caused by the prior task prerequisites, since these tasks can only be done once the others are completed. Below you can see some of the possible orders we got.

```
[cutting meat, heating water, cutting onions, cooking meat, frying onions, cooking potatoes, cooking everything]
[cutting meat, cutting onions, heating water, cooking meat, frying onions, cooking potatoes, cooking everything]
[heating water, cutting meat, cutting onions, cooking meat, frying onions, cooking potatoes, cooking everything]
[heating water, cutting onions, cutting meat, cooking meat, frying onions, cooking potatoes, cooking everything]
[cutting onions, cutting meat, heating water, cooking meat, frying onions, cooking potatoes, cooking everything]
[cutting onions, heating water, cutting meat, cooking meat, frying onions, cooking potatoes, cooking everything]
```

This gave us the idea, that instead of testing all possible orders, we should firstly check which tasks are required to be done first and put those at the start of the list. Furthermore, the ones, who are required for more tasks, should be done first, meaning put at the start of the list, and others, which are not required for any task, should be left behind. With this in mind we wrote the function `smart_permutations`, which takes the name of the recipe json file and returns some permutations of the tasks, where we know one of them will give us the best order. Here we can see the simplified process:

```
- get tasks from json file
- for task in all tasks:
    get task prerequisites (can be repeated)
- count repetitions of tasks
- sort in lists based on number of repetitions
- get all other tasks
- permutate each task list separately
- make all possible combinations
```

This is afterwards inputed into the function `run`, which tries all the given orders and returns the one with the shortest duration.

2.4 Introducing randomness

Since cooking does not always go as planned, we added some randomness, by prolonging/shortening the duration of tasks. We started by defining a parameter `time_change_factor` at the start of the code, which states for maximum, how many 'seconds' can a task be prolonged or shortened. This is all done in the function `time_randomness` that is called when scheduling a task and calculating the time it will be completed. The following graph of our simulation is on figure 2, where the function f represents the previously mentioned function.

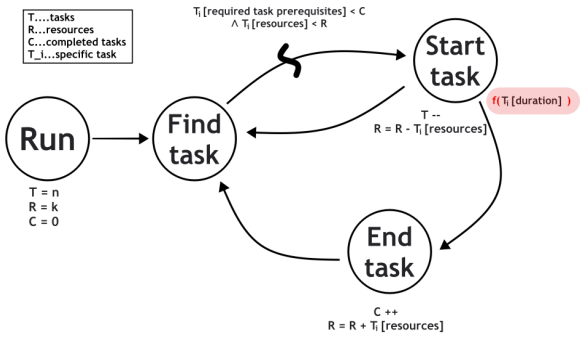

Figure 2: Discrete simulation graph of a cooking process with time randomness.

3 Simulation Results

For the simulation results we looked into a few different problematic topics.

3.1 Effect of different ordering

We wanted to see how big of an impact has the order of the tasks. That is why we looked for the best order of the recipe for goulash, and the worst one. Meaning the one that is done the quickest and the one that takes the longest. The following results are done with two cooks and time randomness.

```
BEST ORDER: 168,
[cutting meat, peeling potatoes, heating water, cutting onions, cooking meat,
cooking potatoes, frying onions, cooking everything]

WORST ORDER: 253,
[cutting onions, cutting meat, heating water, cooking meat,
 frying onions, peeling potatoes, cooking potatoes, cooking everything]
```

We can notice that the difference in duration is 85 units of time, which is not small. This happens due to probalistic duration of time. Even without that, the difference would still be 60 units. The reason for that is the order of the first few tasks, which have to be done before cooking potatoes and cooking everything. If they are not done in the best order, the simulation has to wait for a task to be finished before doing anything new.

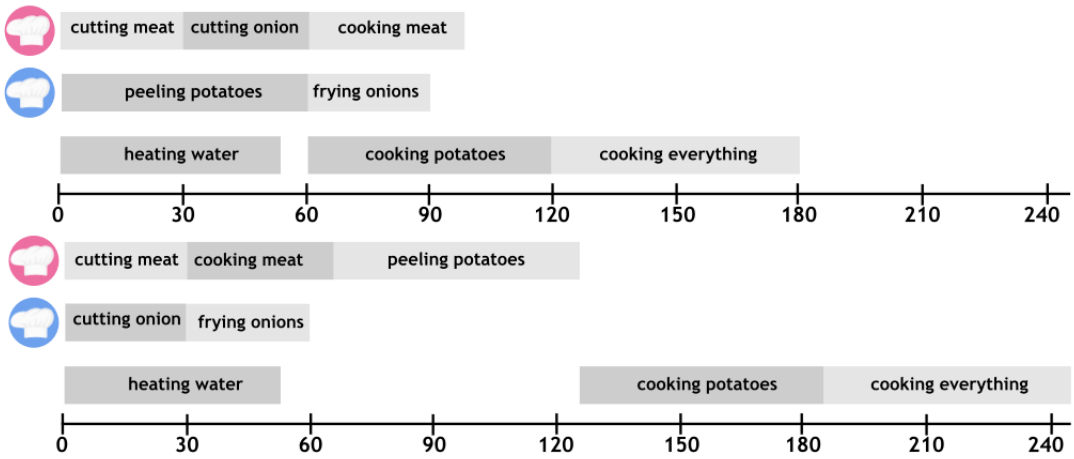

Figure 3: The best and worst order of tasks.

After seeing the difference an order of tasks can have, we were interested to see what are all possible durations of the recipe and which are the most common. In the histogram 4 below we can make out that most of the time the recipe is completed in between 176 to 186 units of time.

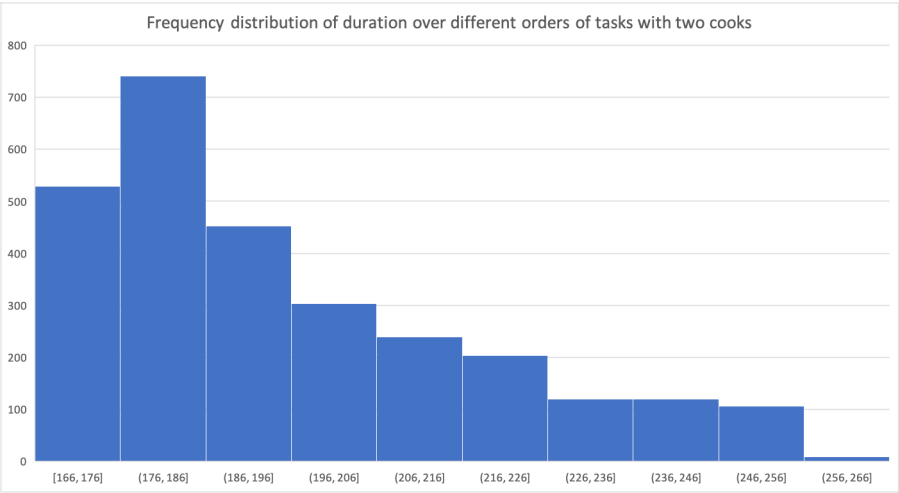

Figure 4: All possible durations of the recipe with two cooks, devided into groups by time interval 10 units.

We also did the same calculations for just one cook, but got a somewhat different distrubution, with most of the orders gathered in the middle, and there is not one interval that would significantly stand out. Most orders of tasks were completed in 240 to 260 units of time. This corelates with the optimal time for one cook, which is 251 units of time.

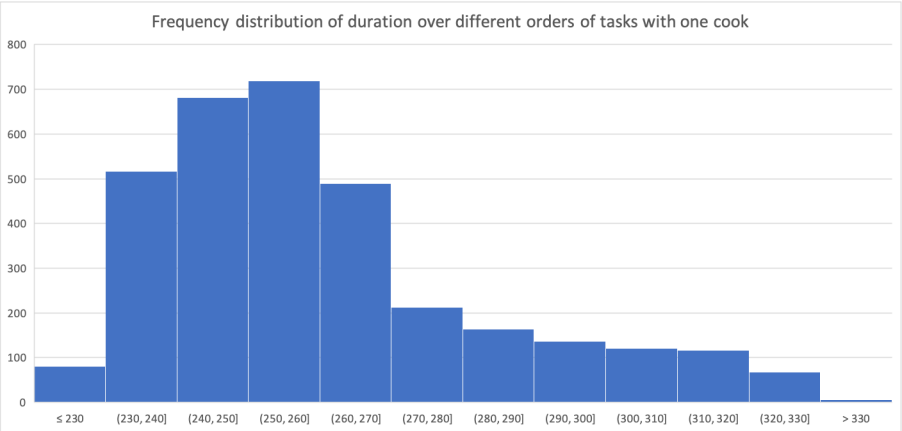

Figure 5: All possible durations of the recipe with one cook, devided into groups by time interval 10 units.

3.2 Recipe comparison to cook book

For our second dish we decided to do potato salad. We found a recipe in a cook book [1, page 42] and adjusted it to our program. We wanted to test if the calculated optimal order for one cook matches the order in the cook book. Here we can see the two orders:

```
COOK BOOK:
- cooking potatoes
- cooking eggs
- cutting pickles
- cooling eggs
- cutting onions
- peeling eggs
- cutting eggs
- peeling potatoes
- cuttting potatoes
- make dressing
- mixing enerything
```

```
OPTIMAL ORDER:
- cooking potatoes
- cooking eggs
- cutting onions
- make dressing
- cutting pickles
- cooling eggs
- peeling eggs
- peeling potatoes
- cutting potatoes
- cutting eggs
- mixing enerything
```

We can spot that the two orders do not differ much. Some tasks are done in a different order, but those are mainly interchangable. One different thing is when the dressing is done. This is because in real life you would want it to be freshly done before completing the salad. Unfortunately, our program does not take this into account.

3.3 Different number of resources

Lastly, we focused on the number of resources to see if it really has that big of an effect on duration. We ran the program 200 times for each number of cooks and took the average of the optimal time. We gave the other resources a high enough quantity so that it will not affect the duration. These are the graphs we got for the recipe goulash 6 and for potato salad 7.

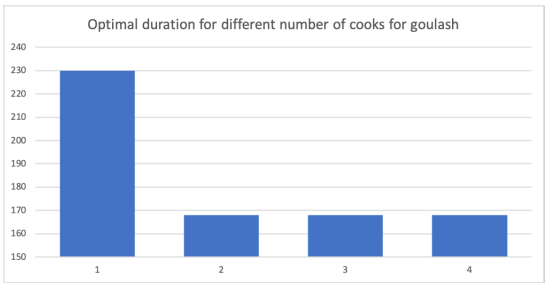

Figure 6: Duration of recipe goulash with different number of cooks, calculated average over 200 iterations.

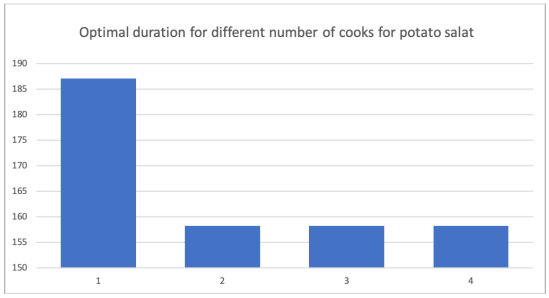

Figure 7: Duration of recipe potato salad with different number of cooks, calculated average over 200 iterations.

As you can see, there is only a difference between using one or two cooks. After that you can use as many as you want, but the duration will not change. This happens because the tasks in our recipes already have an order in which they have to be done. Even if you have another cook, he still has to wait for the other task to finish. To challenge our program we decided to make another recipe, where we tried to construct it so, that a different number of cooks will have a bigger effect.

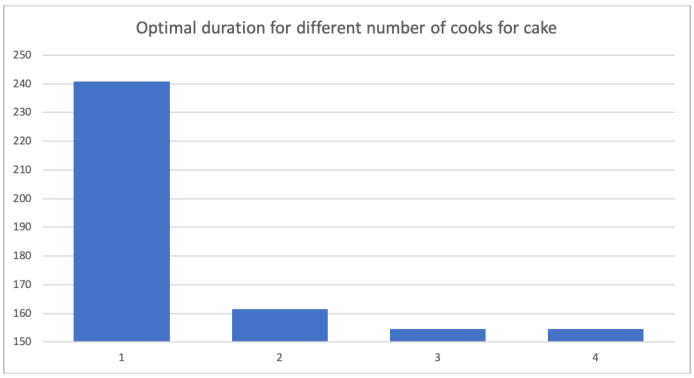

Figure 8: Duration of recipe cake with different number of cooks, calculated average over 50 iterations.

These 8 are the results we got with implementing the recipe of baking a cake. We wanted to have more independant tasks and less strict prior ordering. In the recipe you have to do a lot of tasks independantly and just a few tasks that combine all of them. This way you can spot a difference between one, two and three cooks. However, when it comes to four, the duration still remains the same. Furthermore the difference was very small, only 6 time units. This brings us to the conclusion that with every recipe there will be a point where the number of cooks does not matter anymore.

Besides observing the effect of the number of cooks, we did the same for other resources. Following you can see the duration for making goulash when having:

- double of everything: 181
- only one pan: 191
- only one knife: 181

This happens because, as you can see on figure 9, in the optimal recipe two knives are never used at the same time, while the pan is used for cooking the meat and frying the onions which is done at the same time.

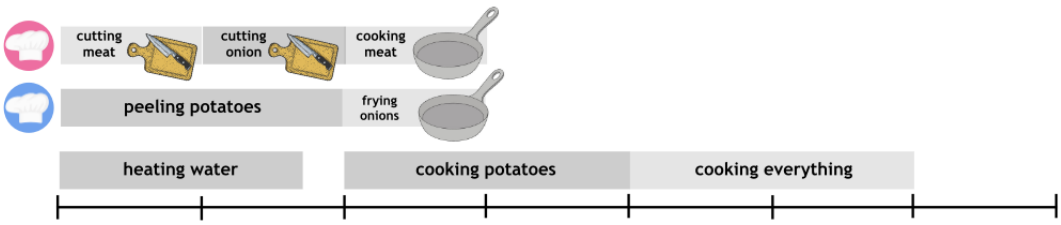

Figure 9: Use of pan and knife in the optimal order of the recipe goulash.

4 Discusion

When we were first introduced to the problem, our understanding of it did not match the supervisors. That is why our first model, done with SimPy, turned out to not be the best solution. In the program our recipe was executed in real time, but the model only did the tasks in the given order, whereas the task given was to find the optimal order. We then decided to do the whole project again from the start. This time without SimPy and with a stronger emphasis on a discrete event model.

Our current model works with three different recipes, goulash, potato salad and cake. Each of the recipes was done with a purpose to further improve our model and test its abilities. We learned that there are recipes, where the order is not that important, and therefore are not the best to use in this case. One example is a fruit salad. But on the other side, having a recipe like making Wiener schnitzel, that already has a very strict order, regarding what tasks have to be done before the other, is not the best option as well.

Our main question of the project was wheather the order affects the duration and if there is an order of tasks that will be quicker than the others. This depends on what kind of recipe you have and how many cooks there are.

Lets starts with only having one cook. In this case there can be a difference in duration, but only if we have tasks that can be done without the cook present. For example in the goulash recipe, if the cook peels the potatoes more at the start, he does not have to wait for them to finish cooking and can instead do something in the mean time. However if he does this task at the end, then he has to wait for them to cook before cooking everything. If we compare this to the potato salad recipe, we can see that there we is no difference in time. That happens beacuse the only two tasks that can be done without the cook present can be done at the start, since there are no tasks required before. The order of the remaining tasks will then not change the duration.

It is very similar when it comes to having two cooks. In the recipe goulash, the optimal order is to start with peeling the potatoes, so that cooking them can be done in the background while the cooks are doing something else. We realized that in the potato salad recipe there is only one possible order for our program to execute the recipe with two cooks. This is because all tasks regarding onions and potatoes take long, and have a specific order. We have to keep in mind that our model will always, if possible, do the tasks that don't need the cook present. This eliminates a lot of possible bad orders, that a person could do in real life. That is why it starts with cooking eggs and potatoes and will never make the mistake of doing that later and having to wait.

Does having more cooks alway pay off? That was one of the other questions we asked ourselves during the simulation testing. As seen in the results sectin above, having more cooks only affects the time to some extend. There usually is a difference between having one or two, in our case the duration decreased by half. But when it came to three cooks, the difference was minor if there even was one. We also have to take into account that even when having more cooks, you also need more pots, pans, knives... While running the same test for potato salad, having only one of every resources, the difference in quantity of cooks did not affect the duration. We also tried executing the recipe goulash with different number of kitchen utensils. There we observed that with one pan the duration changed but with only one knife it was the same as the optimal duration. This is because in the optimal order two pans were used at the same time, whereas the knives were never used simultaneously. This brings us to the conclusion, that big kitchens, or factories, should only have more workers, if they can give them the tools to work with and know

that their work is not too dependant on other work being completed. For example employing more servers, but not having enough cooks. This way the waiters would have to wait for the dishes to be done, and the duration would not decrease.

Lastly we think our model could still be improved. We could add another parameter, where you could say wheather you want this task done last before finishing, or if some task has to be done right after it. This was also visible in the comparison we did to the real recipe for potato salad. In the cook book's case the dressing was done last, since it has to be fresh. But our simulation returned an order where it was done more at the start. Another thing that could make our model more realistic, would be to change the tasks that do not need the cook present. While we simplified the process by stating that the cook is not needed throughout the entire task, there are small tasks at the start and end that require the cook's attention and availability. For example cooking potatoes. It is true that the cook does not have to be present while the potatoes are cooking, but he still has to put them in a pot and take them out at the end. We beileve this could be done by deviding the task cooking potatoes in smaller tasks and adding a stricter ordering, as said before.

References

[1] Cook book. (n.d.). Simply Recipes. Webside: https://www.simplyrecipes.com/dinner-recipes-5091433.

[2] Cooking Recipe Simulation Assignment.

[3] Lectures from subject Modelling and Simulation, version for the academic year 2023/24.