Chair for System Simulation

Winter Term 2021/2022

Prof. Dr. Harald Köstler, Jan Hönig, Marco Heisig

# Advanced Programming Techniques
# Sheet 2 — Tipps and Tricks for the Factorio Build Order Optimizer

Implementing the Factorio Build Order Optimizer (FBOO) might seem like a big task. And it certainly is, especially if you compare it to other programming exercises given at the university. In the FBOO sheet, there is a set of exercises, which might lead you in the right direction. However, to take the edge off for the less experienced students, this sheet will include resources, which might help you with specific and general tasks.

## 1 Setup

This section describes the initial setup of your FBOO project.

### 1.1 Git

A version control system (VCS) is crucial for tackling projects of this size, and is independent of the team's size. Even using a version control system alone is a great habit. Sadly, VCSs, among other things, are rarely stressed in most lectures. This is even more mindboggling when supervisors of students' bachelor's and master's thesis and projects are annoyed when the students don't know `git`. Luckily VSCs like `git` are heavily used in the industry and many projects, so a wide selection of tutorials are available.
We give you at least some string points for learning `git`[1]:

- gittutorial
- The Book
- learngitbranching
- git videos
- Atlassian Tutorial

After you have some rudimentary knowledge about `git`, you need to have a repository somewhere online to exchange your work with your teammates. You can choose whatever you want, as long as your repository remains *private*, so only your team can access your code[2]. We recommend using the FAU Department Informatik GitLab. However, feel free to use GitHub, GitLab, Atlassian, SourceForge, or your own `git` server.

### 1.2 Build tool

Typing `g++ -std=c++17...` manually for every file for every compilation is a... little bit tedious? A build tool can do it for you. There are tons of different build tools, which support C++ (Meson, Gradle, Ant, Make, your solution). We recommend `CMake`[3]. It is a de facto standard, which many C++ projects use.

[1] `git` is not the only VCS available. `svn`, `mercurial` and other alternatives are feasible as well. We recommend using `git` since it is the most used VCS.
[2] It is not our problem if due to plagiarism you won't receive bonus points!
[3] If you would like to use anything different. Please make sure that there is a good deb package, and *tell* us so we can install it on the validation server.

`CMake` itself dosn't build anything. It creates configuration files for other build tools like `make` or `ninja`, which then at last compile and link your code. `CMake` supports the generation of configuration files for various backends.
Here are some introductions to `CMake`:

- An Introduction to Modern CMake
- CMake Tutorial
- Effective Modern CMake

Don't forget that `CMake` is coding as well, and it definitely shouldn't be just some half-baked solution.

### 1.3 Project Structure

Unless you want to implement everything into one file (which is possible, just usually not a very good solution), you need to have some pattern where individual files go. We like the proposal in How to structure your project. Usually, you will find a handful of different folders and files in a project like FBOO[4]:

```
- project
  - .gitignore
  - README.md
  - CMakeLists.txt
  - include
    - project
      - lib.hpp
  - src
    - CMakeLists.txt
    - lib.cpp
  - apps
    - CMakeLists.txt
    - factorio.cpp
  - extern
    - googletest
  - scripts
    - helper.py
```

## 2 Engage!

After you *carefully* read the sheet with the task and set up your project, repository, and environment, you may wonder "What now?". Classically you would first analyze the problem and then create a design of the software[5]. However, this presumes that you have experience in design and programming. There are steps you can do before you spend too much time on designing the FBOO. Splitting the task into multiple smaller tasks (divide and conquer) helps you tackle this project step by step while learning on the job.

### 2.1 Slay the JSON

First things first, read in the json files `factory.json`, `item.json`, `recipe.json` and `technology.json`. Think about how you want to represent the information contained in those files. Think ahead about how you want to use the information. C++ gives you a lot of options (tuples, classes, vectors, sets, maps, strings,

[4] We omitted `tests` and `docs` because we know you :)
[5] Software design

enums, ...). In any case, we recommend using the nlohmann/json library[6]. It is also possible to implement your specific JSON parser.

Read in the *target* file as specified. Here you should already think about how to represent the current state of the simulation.

[6]This library is installed on the validation server and in CIP-Pool.