1 Concept

My solution is a flask-based webapp. A request has the structure URI/troops?size=[number], where *number* can by any unsigend integer >= 3. The service response is an application/json string. In case of an unknown endpoint, the service returns an http_404 and in case of an illegal request parameter an http_400 - both also in application/json.

It is build with the MVC pattern and easy expandability in mind. Any further *character class* such as Cavalry or Trebuchet could be added to the */application/app/config/settings.py* document and the programm will behave accordingly. Any other data could be added as a member to the *Army* class as long, as it inherits from the *JSONable* base class.

2 Deployment

Prerequisites A unix-based host with the Docker engine installed.

Deployment In the project folder is a shell script *deploy.sh*, which builds a docker-image and subsequently launches it in detached mode with a docker-compose command. Everything is ready configured and the service is available under *localhost:8080* after launch.

Deployed Service An instance *https://gg.juliusneudecker.com/* is already available as a deployed version. The tests partly use this URI as a request endpoint.

3 Structure

The main entrypoint for the application is the *__init__.py* file in */application/app/*. Apart from some configurations are three handlers for http handling: *index, apiHandler, error*. Where *index* serves the landing page and *error* are the error handlers in the project for the 4XX-Errors. The *apiHandler* responds to the */troops* endpoint and builds the response by creating an Army object with Troop children, where the *count*-property of each child-Troop object is determined by the algorithm in */application/app/functions/randomNumberProvider.py* - which is the solution of the core problem of this challenge. The Army-Object is subsequently parsed into a json-string and put into the http-response.

4 Testing

The core-modules for the program each have their dedicated unit test. In the project directory under */application/app/test/*. To execute these tests with python, you have to create a venv with the packages specified in */application/requirements.txt*. Then every unit-test can be launched seperately.

There is also the *evenDistributionTest.py*, which requests a certain number of datasets (set to 1000) and makes a boxplot for every unit-type, to show that there are is no bias in the algorithm.