# The Flood People Simulator

## Introduction

The Flood People Simulator is a simulator for dynamic modelling of interactions between flooding and people in crowded areas. It can be used to simulate flood events in 3D spaces such as sports stadiums, shopping centres and residential areas. The Flood People Simulator is developed in FlameGPU (Flexible Large Scale Agent Modelling Environment for the GPU) which is a framework used to build agent-based models which can use GPUs for their processing power.

Both FlameGPU and Flood People Simulator are explained in more technical detail below.

## Overview of FlameGPU

FlameGPU is an extension to a relatively well-known tool in the modelling community called FLAME. Models created in Flame are created based on a computational model called finite state machines. By unifying the way that models are created, we ensure that programs can be generated and they scale and run well for High Performance Computing (HPC). FlameGPU works in much the same way as FLAME and allows users to specify their models using XML format. As well as the XML file, users must include at least one function file which is a source code file containing the agent functions for the model.

Users wanting to use FlameGPU to produce their model must first fork the repository from GitHub and then add their model on top of the code.

## Overview of Flood People Simulator

The Flood People Simulator model itself is built on top of the FlameGPU package. It has been configured to take in two XML files. One XML describes the 3D environment in which the flood event will take place. This takes the form of a series of x, y, z coordinates which build up a 3D space. The documentation for the Flood People Simulator suggests using FlameGPU's FGPUGridNavPlanEditor (https://github.com/RSE-Sheffield/FGPUGridNavPlanEditor) to put this file together. The second XML file describes a set of parameters that affect the metadata of the flood (i.e. the flow rate of the flood, the speed and number of pedestrians, whether sandbags are available).

The simulation itself can be run in either "visualisation" or "console" mode. Either mode will simulate the flood step by step and, at each stage, will output the number of survivors as well as other information about the flood itself.

## Analysis of Software

Both FlameGPU and the Flood People-Simulator are written mainly in C and C++ and are built using makefiles and configured using XML files. They are also both open source and available at the following links:

- FlameGPU – https://github.com/FLAMEGPU/FLAMEGPU
- Flood People Simulator – https://github.com/SahebSh/FLAMEGPU

As mentioned above, once built using the makefile, The Flood-People Simulator can be run in either visualisation or console mode. For DAFNI purposes, we will be running in console mode as the model will not be running on an operating system which has a GUI to display the visualisation.

## Input Data

The Flood People Simulator doesn't have any dependencies on external datafiles to run. It only requires the 'initial_state.xml' file which outlines what the 3D space looks like and the 'map.xml' file which contains all of the configurable metadata about the flood itself.

## Configuration

There are many configurable parameters for the model. The main ones are listed in the table below.

| Parameter | Description | Format | Unit | Value ranges |
|---|---|---|---|---|
| *freeze_trapped_peds_on* | (de)activate immobility of pedestrians (Shirvani et al. 2020) | integer | **none** | 1: Activate<br>0: Deactivate |
| *ped_roughness_effect_on* | (de)activate the effect of pedestrians' bodies on bed roughness. | integer | **none** | 1: Activate<br>0: Deactivate |
| *sandbagging_on* | (de)activate the intervention case prior to the start of flooding at a specific time (see further below). | integer | **none** | 1: Activate<br>0: Deactivate |
| *pedestrian_population* | To choose the maximum number of pedestrians in the area at the start of flooding. | integer | **none** | Any positive value |
| *hero_percentage* | To define the percentage of emergency responders (of total population) involved in the intervention case. | float | **none** | 0.01 (1%) to 1 (100%) |

| | | | | |
|---|---|---|---|---|
| *inflow_start_time*<br>*inflow_peak_time*<br>*inflow_end_time* | To define the inflow condition in terms of time according to the hydrograph shown in **Figure 5a** (Shirvani et al. 2020). | float | **second** | Any positive value |
| *inflow_initial_discharge*<br>*inflow_peak_discharge*<br>*inflow_end_discharge* | To define the inflow discharge according to the hydrograph shown in **Figure 5a** (Shirvani et al. 2020). | float | **m3/s** | Any positive value |
| *evacuation_start_time* | To specify the time of early evacuation (*evacuation_on* should already be activated). | float | **second** | Any positive value |
| *sandbagging_start_time*<br>*sandbagging_end_time* | To specify the starting and ending times for the intervention case (*sandbagging_on* should already be activated). | float | **second** | Any positive value |
| pickup_duration<br>drop_duration | To specify how long each emergency responder may spend on picking up and dropping a sandbag. | float | **second** | Any positive value |
| *sandbag_length*<br>*sandbag_height*<br>*sandbag_width* | To specify the dimensions of each sandbag. | float | **meter** | Any positive value |

| | | | | |
|---|---|---|---|---|
| *pickup_point* | To specify the location of the sandbag storage, e.g. truck (see **Figure 4b** in Shirvani et al. 2020). | integer | **none** | 1 to 10 |
| *drop_point* | To specify the location where emergency responders are asked to drop the sandbags (see **Figure 4b** in Shirvani et al. 2020). | integer | **none** | 1 to 10 |

## Output Data

The output data consists of a set of CSVs. A CSV will be produced for each timestep in the simulation which will list the current state of the 3D space at that point in the timestep which includes the contents of each cell (flooded, not flooded, how many people in it, is it sandbagged etc…). There is also an overall CSV outputted at the end of the simulation to give some overall information about the impact of the flood event.

## Pilot Scope

### Objectives

The main aims of the pilot project were to:

- Containerise FlameGPU separately from the Flood People Simulator so that other FlameGPU based models can be more easily ported onto DAFNI in future
- Containerise the Flood People Simulator using the FlameGPU base image produced in the first step
- Write the model schema for the model indicating which parameters can be changed and specifying minimums / maximums
- Write a wrapper around the FlameGPU code which takes in environment variables and writes them to the appropriate location in the ‘map.xml’ file allowing DAFNI users to easily change the parameters.

### Challenges

Getting both FlameGPU and the Flood People Simulator dockerised was a fairly trivial process. Both codebases were solid with good documentation and the owners of Flood People Simulator were very responsive in dealing with any questions or problems we had.

One challenge was working with the C code. Up until now, we haven’t had any DAFNI models which are coded entirely in C. We also don’t have any members of the team that are experienced with C which meant that the model itself was quite difficult to debug. We ended up spending a while in adding a lot of extra logging to the main files to figure out where things were going wrong. This eventually led to me discovering a bug in the source code that

we then fed back to the developers. This bug was then fixed and we were able to progress in getting the model uploaded onto DAFNI. This really highlighted the importance of being able to see logging output from the models as they run and how much someone might struggle if they didn't have access to those logs.

Another challenge in this pilot was enabling GPUs to be selected as a resource in a model definition. In the early stages of development, we were manually submitting jobs to the DAFNI Kubernetes cluster and was able to configure the job to use GPUs manually in my configuration file. When it came to actually running the job in Kubernetes, we had to do some extra configuration to get the GPUs working in the Kubernetes nodes. In addition to this, when running through the NIMS we had to add an additional parameter to the model schema file to allow a model to be "GPU-enabled".

Overall, the challenges in this pilot were relatively few. This is testament to the documentation of the codebase which was extremely useful and all encompassing.

## Outcomes

The outcome is that the model is now running on DAFNI and is configurable by the user in the workflow creation part of the web application. There is a remaining piece of work to do to allow users to upload their own 3D space XML file. At the moment, there is no way of configuring this, so the flood event always runs on the same example 3D space file that is included with the model. This is obviously a significant limitation, but it will be a very easy fix once we have completed the "Select dataset for model" feature on DAFNI which is going to be worked on in upcoming sprints.

## Lessons Learned

- Trying to get the model to run without any debugging output highlighted the need for technical logging being made available to users throughout the model upload and workflow execution process. Without these logs they will have no hope of fixing models when they go wrong.
- Learned a lot about GPUs being used on DAFNI including doing some stress testing to see how Kubernetes handles an excess of GPU jobs. Also, all of the work to run GPU-based models on DAFNI has now been done so this will make future GPU-based models much easier to port onto DAFNI.
- Used this model as the first full test for uploading a complex model through the model upload process. Also we followed the guide as if we were in the place of a first-time DAFNI user. Learned that there are some gaps in the documentation around wrapping the Flood-People simulator to use environment variables amongst other things.