University of Sheffield

A tool to specify testable causal models of software behaviour

Hao-Hsuan Teng

Supervisor: Dr Neil Walkinshaw

COM3610

A report submitted in fulfilment of the requirements
for the degree of MSc in Advanced Computer Science

in the

Department of Computer Science

December 4, 2021

Declaration

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Name : Hao-Hsuan Teng

Signature : Hao-Hsuan Teng

Date : 12/4/2021

Abstract

With the inventions of supercomputer with powerful processing power, a common way to utilize it is to run complex computational models that can be used to predict the outcome of a certain event. These modules often involve large number of inputs and outputs and can be extremely time consuming to test. The goal of this project is to create a tool that can be used to test these complex computational models, this tool needs to be easy to understand, and accessible for people who have limited software testing experience. Parts of the tool for this project has already been in development, but it still requires some functions to make the system user-friendly. A simple user interface with the ability to display results and create new test sets have been implanted, the results show that this system is capable of assisting non-software engineer users access this system without too much trouble.

Contents

List of Figures

Chapter 1

Introduction

Throughout the years, companies have developed CPU with more and more powerful processing power, and these more powerful CPU comes the invention of supercomputer, these supercomputers have superior computational power compared to normal computers. With this computational power, people start to develop large and complex computational models. Computational model is a way to try and use computer system to simulate real life situation, these models use large amounts of parameters to determine the outcomes of a certain event, researchers can utilize these models to simulate experiments and real-life event [1] and aid their research with the results.

But these models aren't without problems, one problem is these models has a large number of inputs and outputs, and it takes a lot of times to process them, this means in order to test them, it requires a lot of time to test all inputs and outputs' accuracy with traditional method.

The solution provided by this project is to use the causal models, by determine the relationship between different input and output parameters and use this information to create graphs that focus on certain behaviours. Then just simply compare the difference between the test sets and the output of the tested models, people can know how accurate a certain interaction is between input and output parameters, then with these result, testers can decide whether or not this system meets their expectations. By using this method tester can saved a lot of time compared to traditional method where the best ways to test it is to run the model repeatedly with new data [11].

1.1 Aims and Objectives

Despite the existence of the theory, there isn't a software where the user can store these test sets and used these sets to test other tools. So, primary goal of this project is to provide a system that can be used to store the test sets, and when needed, people can specify the

scenarios they want and use the stored sets to test other computational models.

This system will be primary based on a tool called Causcumber, it is a tool for testing computational model and is mainly based on a model called Covasim, Covasim is a computational model focus on COVID-19 analyses, it can determine the infection rate and how will the rate change based on different interventions (such as quarantine, social distance, etc.) [7]. Causcumber is a in developing tool and is aimed to provide a testing method for computational model like Covasim, it uses Cucumber specification, another tool reads executable specification in plain text and validates if the software does what those specifications say, to produce casual model graph.

The main problem with Causcumber is that it lacks an accessible user interface, so for users who haven't really studied software engineering, it can be difficult to understand. So, the goal is to eventually provide a user interface that can be accessible for all levels of users.

Another goal of this system is that it needs to be easy to read, and to achieve this, we use gherkin reference to help with it, by using a set of special keywords with a certain structure, it can give meaning to executable specifications. For the users who are not specialize in software engineering, this is a way to make the system easy to understand and can avoid a lot of confusion.

1.2 Overview of the Report

In the next chapter, Literature Survey, a list of literature will be providing information and reasons why there's a need for this type of testing tool to exist. It will start with explaining what scientific software is, what they do and why they aren't usually tested in a proper method. Then this will fellow up with introductions to Cucumber and Behave, two of the main factors of the Causcumber system. After that will be explain what is Causal testing and how computational model testing could benefit from it.

In the third chapter, Requirements and Analysis, will be mainly talking about the requirements and objectives of the system. And the design choice was made based on these requirements.

Part four, Progress, will be discussing the current state of the system, how it is received and what feedback it gets.

In the fifth chapter, conclusions and project plan, will summarize what has been achieved till this date, and the overall plan for the upcoming months.

Chapter 2

Literature Survey

This section will explain why there's a need for this system by exploring background literature related to this subject. Mainly the need of a proper system to test scientific software, start with explaining what a scientific software is, following up by explaining why most of them are not tested correctly. Then we will be discussing the two main components of this system, Cucumber and Behave. Then finally, explaining what casual testing is, the main method used by this system to test other software, and exhibit why it lacks a proper interactive system for non software engineering users.

2.1 Scientific software

Using a computational model for scientific research is common practice with today's technology, scientists use these models to predict the result of an experiment or the outcome of an event. But whether these models can accurately predict the result, is still a problem since most of them lack some form of proper testing. Below will be discussing what is a computational model and why they are poorly tested.

2.1.1 Computational model

What defines a scientific computational model, it's by using computers to simulate and study real life events by using mathematics, statistics, physics, and computer science to study the mechanism and behavior of complex systems by computer simulation [1].

By using models that contain a number of input variables, and algorithms that will define the system. These models will simulate real-life situations and researchers can adjust these models by changing their variable and algorithm according to the results to make the model more conform to reality. Then researchers can use these models to see how one or more variables can affect the outcome of a certain event. Computational models provided some level of prediction of being able to calculate an anticipated result from a given set of variables [10]. This forecasting system can be used to predict complex systems, such as weather or

disease spread, and help researchers or decision makers to decide their next move.

An example of this kind of model is Covasim [4], Covasim uses agents to simulate individual people, the model mainly focused on one type of calculation, what is the probability of an individual in a given time step will change from not infected to infected, or badly ill to death.

The simulation starts from loaded the parameters, then it will start creating individual with different age, sex, and comorbidities based on the selected location (i.e., Different country). Then will be grouped into a social network depending on their attributes. After that the model will start looping, in each step, the model will apply various operations on the individual, then collect the result and apply analysis.

2.1.2 Why computational models are hard to test

Computational models are often very complicated, and require special knowledge related to the field. These models are often developed by scientists themselves, but most scientists aren't software developers and may not have knowledge in some common software engineering practices, this may cost the quality of the scientific software [13]. Software testing is one of the aspects that is impacted, since the lack of knowledge in software development, lack of understanding in systematic testing is expected. Mistake can be made without notice and may affect the output of the system, causing the result to become inaccurate.

Testing a scientific software itself can be a difficult task to do, this may be the result of two types of challenges:

First is due to the software's main purpose is to predict something unknown or simulate areas even researchers have little knowledge in, but to test a software, it is crucial to know how the software should Behave, what kind of output should it show. Without these knowledge, it is hard to tell if the software is working the right way.

Furthermore, these computation models are often consisting of hundreds of parameters and complex algorithm, this means it would require lots of test case in order to test every aspect of the system, and this leads high execution time making the testing process become very time consuming [13].

Second is that scientific software is mostly developed with scientists being the lead role instead of a software engineer, and the value of the software system is often underestimated [13]. And most scientists never went through the training in software design like software engineer [12], this means limited understanding of the testing process and not applying known testing methods. This will make the core design of the software unfriendly for effective testing.

2.2 Current testing method

Despite the difficulty in testing these models, there are still some common software testing methods that can be used to test them, such as behavior-driven development (or BDD) and Cucumber. Combined with casual model testing, a way to draw relationships between variables, it will make testing these complex models a lot easier.

2.2.1 Behavior-driven development with Behave

Behave is a python API based on behavior-driven development, behavior-driven development is a development technique that encourages collaboration between participants in a software project [2]. Behavior-driven development or BDD, aims to have a clear understanding of the desired software behavior between stakeholders and software developers, and communicate by writing test cases in a natural language that both sides can understand. Behave is a Python API created for this, it consists of tests written in Cucumber, an API written in natural language style.

When a test case is written, it requires a tool to read its specifications and see if the system works as the test-case specified. Cucumber is the tool that was developed for this purpose, it reads specifications written in plain text with a certain format and validates if the software does what those specifications say [8] and creates a report.

For Cucumber to understand the specifications, it must be written in Gherkin language, Gherkin uses a set of grammar and special keywords to give plain text structure so that Cucumber could understand it, one of the pros for Gherkin is that keyword can be translated to other languages, making it usable for people who don't speak English [9].

A proper Gherkin grammar starts by giving a context, then describing an event, and after that what should happen after the event. With these grammars, testers can specify a situation in the system, describe a certain parameter, and what the results should be with those parameters. By writing a test using Cucumber and Gherkin grammar, tester can combine them into a .Feature file as display below:

```
Feature: Compare interventions
  Background:
    Given a simulation with parameters
      | parameter     | value      | type |
      | quar_period   | 14         | int  |
      | n_days        | 84         | int  |
      | pop_type      | hybrid     | str  |
      | pop_size      | 50000      | int  |
      | pop_infected  | 100        | int  |
      | location      | UK         | str  |
      | interventions | baseline   | str  |
    And the following variables are recorded weekly
      | variable          | type |
      | cum_tests         | int  |
      | n_quarantined     | int  |
      | n_exposed         | int  |
      | cum_infections    | int  |
      | cum_symptomatic   | int  |
      | cum_severe        | int  |
      | cum_critical      | int  |
      | cum_deaths        | int  |
```

Figure 2.1: A feature file example.

With this .Feature file, tester can execute the file in terminal and Behave can read what needs to be tested and return a report.

2.2.2 Causal testing

With the help of Behave and Cucumber, we have the ability to test computational models. It is possible to test the entire system by writing an enormous feature file, but the execution time will become really long, and since computational models are usually in big scale it is also easy to make mistakes. This is when causal model can help simplify the testing process.

Causal model is a way to represent causal relationship within a system through mathematical model. It helps testers monitor causal relationships in the data [6]. A causal module can predict the behavior of a system, by comparing different inputs and the outputs a system produces, it can explore the cause and influence of a certain relationship [3]. By understanding these relationships, testers can test parts of the system by monitor if the inputs and outputs follow the same behaviour.

With this method testers can reduce the time required to test a complicated system by only testing parts of it one at a time, then combine the results, and testers can have a full picture of how the system performs. This is unlike the traditional method, where it requires testing

the entire system all at once, and may require lots of time if it is a complicated system.

2.2.3 Testing computational model with causal testing

Causal testing can be a really useful testing method for software engineers [5], however computational models are often developed by scientists or researchers who have limited knowledge in software testing [2]. Tester who wants to use causal model as a way to test system needs to have prior knowledge in how variables work in a software, this means people who may not have that much knowledge in software engineering, for example researchers in other fields, might not be able to conduct this method of testing in their software.

2.3 Summary

Behave is a powerful testing tool for software tester's, based on the behavior-driven development method, encouraging people who do not specialize in software engineering to take part in testing. Assisted by Cucumber, a tool where users can describe testing steps in an easy-to-understand manner, and Cucumber compile and produce the results. With causal model, testers can effectively test large and complex computational models, saving time by only test parts of the system one at a time and combining the results together. But the problem is that when using causal model testing, it is inaccessible for most people, due to its lack of user-friendly nature. In order to make it more user friendly, we can integrate Cucumber and Behave with it to make it where users can use causal model testing by using Behave and Cucumber specify the testing process.

Chapter 3

Analysis

Computational models are often too large and complex for traditional methods to test, this often causes developers of those models unwilling to test their models thoroughly. Furthermore, computational model developers are often scientists or researchers that don't have extensive knowledge in software engineering or software testing. Due to the different training received by scientists compared to software developers, the importance of software testing is often overlooked.

Causal model testing can be a powerful tool to test computational model, but causal model testing requires software testers to integrate the causal model with software model that is being tested, consider the complexity of the computational model being tested can be, it is hard to integrate the test for people who don't have software engineering background.

It is possible to integrate Behave and Cucumber to make test computational models with causal model testing easier for people. By integrating Cucumber testers can use easy to understand language to specify the testing process and Behave can read and execute these processes. By combining all these researchers who want to test their computational model can have an easy to access tool to test their software.

A tool with this function has already been in development for a while called Causcumber, a modify version of Cucumber for testing computational model, it has the ability to use Cucumber specification and Behave combine with causal module testing to test computational model, but currently it still lacks an easy to use method and can only be execute in terminal with specific command.

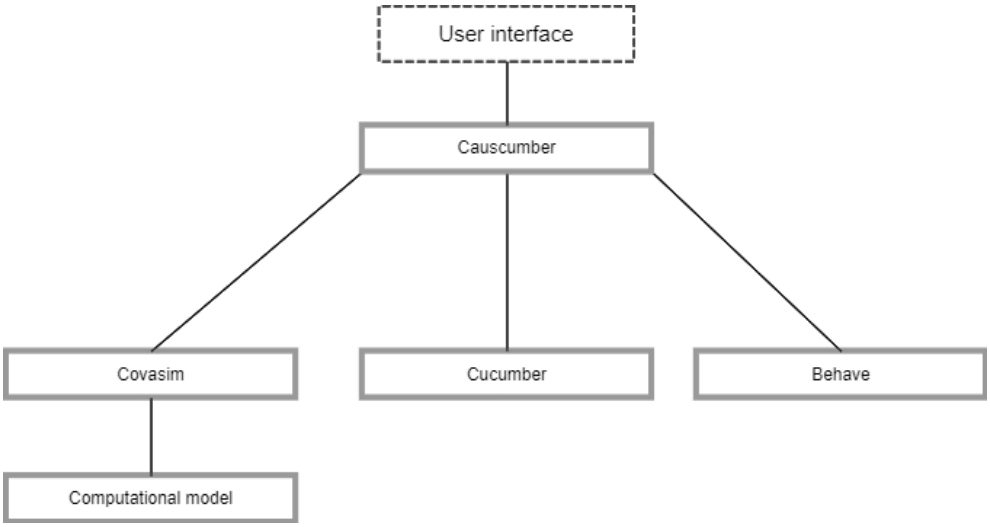

Figure 3.1: A diagram analysis for the system.

3.1 Project Requirements

The main goal of this project is to develop an easy-to-use system where people can use it to execute the feature file they have written, the system will examine the feature file and go through the computational model to check if the model performs as the test specifies. As a result, the system should produce a coherent result detailing the accuracy of the tested model. Thus, the systems need to accomplish the following:

R1. As a user, I want this system to be easy to understand, when I use the system, I want to immediate know what I need to do to get the result I want.

R2. As a user, I want the system to go through the feature file I provide and test the computational model with it.

R3. As a user, I want the system to produce a result where I can know the accuracy of the computational model.

Since Causcumber has been in development for a while, what this project aims to accomplish is adding more to this tool. Currently the state of Causcumber is capable of executing some feature files that are used to test a computational model named Covasim. And it is capable of returning a lot of useful information, but the information still requires some organization to make the result easier to read. And currently there isn't a way to execute the testing system without using the terminal to execute the command, so there's a need for a user interface for people to operate the system. Last is that currently the only way to create a feature file is by hand typing the files, and this might cause some error or confusion. There's also some other requirements according to the feedback of the Causcumber team, these feedback

consist of multiple functions that are potent to the system and will be included during the development. Therefore, by the end of the project Causcumber should be able to accomplish the following:

1. As a user, I want the result produced by the system to be clean and easy to understand, focusing on the important part.

2. As a user, I want to execute and interact with the system through a user interface.

3. As a user, I want to have a more convenient way to create a feature file, to avoid any mistake during the creation of the feature file.

4. As a user, I want to have the ability to toggle different types of result display for different levels of the users.

5. As a user, I want to have the ability to have different input options automatically based on different situations.

6. As a user, I want to have an option to save the feature file I created.

3.2 Analysis

One of the main themes for this project is ease of use, and therefore that's what most of the requirements are focused on. For requirement R 1 – 3, the goal is to make the testing process as simple as possible, with a simplified testing process, people will be more willing to test their computational model.

Since integrating the causal model testing can be difficult and confusing, requirements 3, 5, 6 are focused on simplifying this process, by making this part reduce to only require testers to input the parameters and their value, this should help mitigate a lot of errors.

Finally, this project's main focus is on enhancing the user experience for Causcumber, this is what requirement 1, 2, 4 are for, these requirements make using Causcumber to test computational models a more convenient experience.

3.3 Ethical, Professional and Legal Issues

The execution of this project won't require any real human or animal involvement, every part of this project is simulated, so there won't be any ethical problem. All the source code used in this project is open-source and free to use, the tools used to develop the user interface are also open-source and free to use, so this means there won't be any legal problems either.

Chapter 4

Planning

4.1 Risk Analysis

The main purpose of this system will be providing a tool for people to test computational models. Researchers can use this system to test if the model is working as intended and companies or others who want to use the model can use it to see the model's accuracy. The following are the stake-holder of this project:

1. Developers of the Causcumber project
2. Researchers who will be using Causcumber to test their model
3. Organization that uses this tool to check their model's accuracy

One of the main goals of this model is to return a clear and correct result for the user, if the returned result is incorrect or misleading. This might affect the development of the computational models that use this tool for testing, the result might end up being inaccurate and people who use these results for scientific research or other purposes may be affected by this mistake.

4.2 Project Plan

For this project, I plan to start with understanding how Causcumber works, what kind of result it produces. Then, following by understanding how Behave and Cucumber functions in the system, by understanding how these tools work, it would benefit a lot for future planning. The next phase will be designing the user interface for Causcumber, this phase will focus on the design for user interface and making sure it is easy to understand, and easy to operate. After the design the is done, I plan start implement the system, combine the system with the interface, in this phase will be working in Scrum method, where the project will focus on rapidly adding or change function based on feedback, this is the current phase where the project is in right now, a list of features is either being implanting or will be implement in

the near future.
The last phase will be polishing the system, focusing on debugging, and making sure the system runs smoothly. Below is a Gantt chart showing the plan and approximate time for different phase:

Green: Completed
Blue: In progress
Grey: To do

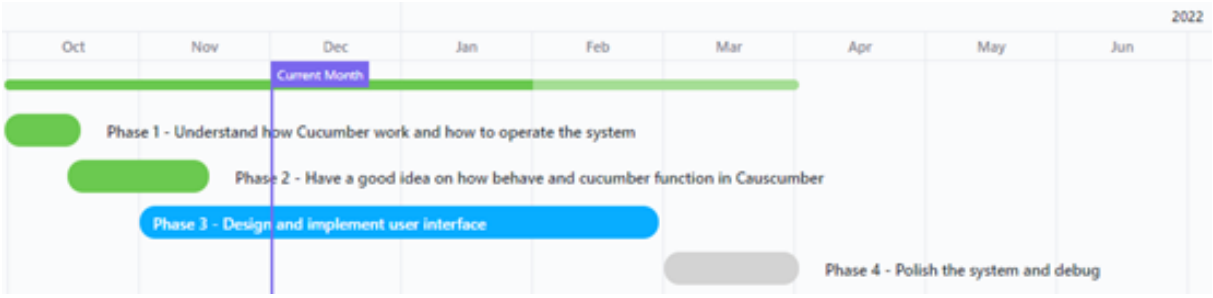

Figure 4.1: Project plan display in gantt chart.

4.3 Current progress

This project already has some progress, I have grasp on how Causcumber work and have move into the scrum phase where a user interface being develop and adding or adjust functions according to feedback, below will be some figure showing the current progress of the system:

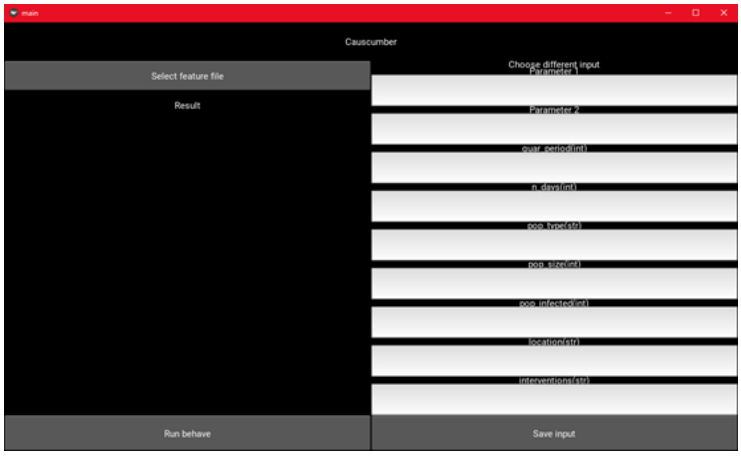

Figure 4.2: An overview of the user interface.

This is an overview of the user interface, the left part will display the result, and the right part is where the users can custom their feature file.

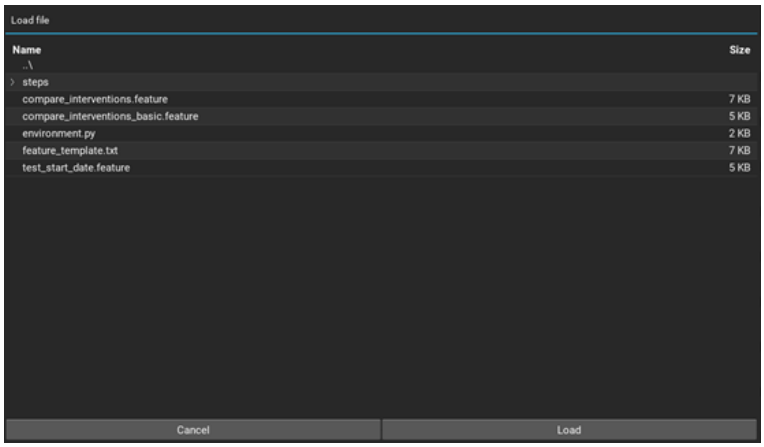

Figure 4.3: Select feature file screen.

User can select which feature file they want to execute, then Causcumber will load and produce the result into a xml file.

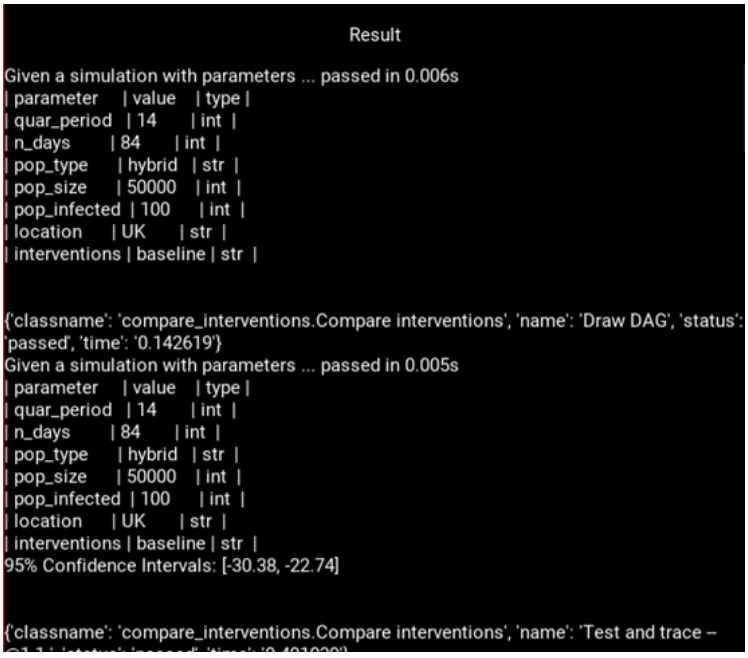

Figure 4.4: Result produced by Causcumber and display in the result section.

After Causcumber loads the feature file and produces it to the xml file, the system can read from the xml file, retrieve the useful information and remove the unwanted ones. Users then can view the input parameters and how accurate a parameter in the computational models are.

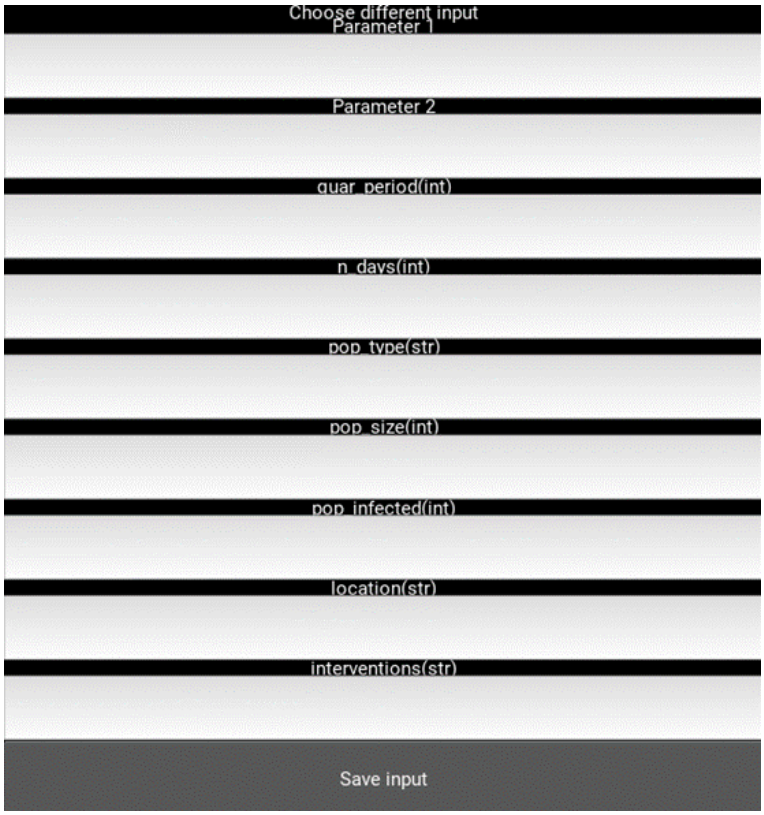

Figure 4.5: Choose different input value, and save it into a feature file.

On the right side of the interface, users can create their own feature file with their own parameter to test the system, the system will compile the input value and save as a new feature file.

Chapter 5

Conclusions

A basic User interface has already been built, with the ability to retrieve useful information and allow users to create feature files with specific value. Users can choose which feature file they want to use to test the computational model, then the system will produce the result and the users can view it. Users can also customize the feature file by choosing their own value and parameter and test the model with their own feature file. In the future, it is planned to add more ways to display the result, allowing users to pick and choose what type of result display they want. Another plan is to add a more flexible feature file create system, allowing people to create feature files with more variety. Below is a detail the plan of work:

Phase1: Understand how Causcumber work

Phase2: Have a good idea on how Behave and Cucumber function in Causcumber

Phase3: Design and implement user interface and focus on scrum to gather feedback and adjust the system according to it

Phase4: Polish the system and debug or add new function according to feedback if needed

*This schedule is subject to change due to the uncertainty in various phase of the project

Bibliography

[1] Computational modeling. *National Institute of Biomedical Imaging and Bioengineering* (May 2020), 1.

[2] Benno Rice, R. J., and Engel, J. Behavior driven development.

[3] Brittany Johnson, Yuriy Brun, A. M. Causal testing: Finding defects' root causes. 123 – 146.

[4] Cliff C. Kerr, Robyn M. Stuart, e. Covasim: An agent-based model of covid-19 dynamics and interventions. *PLOS Computational Biology* (July 2018).

[5] Gaskin, P. B. L. J. Partial least squares (pls) structural equation modeling (sem) for building and testing behavioral causal theory: When to choose it and how to use it.

[6] Hitchcock, C. Causal models.

[7] Kerr CC, Stuart RM, M. D. e. Covasim.

[8] Marit van Dijk, Aslak Hellesoy, e. Cucumber.

[9] Marit van Dijk, Aslak Hellesoy, e. Gherkin reference.

[10] Muffy Calder, Claire Craig, e. Computational modelling for decision-making: where, why, what, who and how. *Royal Society Open Science* (June 2018).

[11] Robert C Wilson, A. G. C. Ten simple rules for the computational modeling of behavioral data. *eLife* (November 2019), 1.

[12] Segal, J. Scientists and software engineers: A tale of two cultures. *Open research Online*, 1 (2008), 2.

[13] Upulee Kanewala, J. M. Testing scientific software: A systematic literature review. *ScienceDirect* (May 2014).