University of Reading

Department of Computer Science
Individual Project – CS3IP16

ZEUS

Zombie Epidemic Universe Simulator

Czar Ian Echavez
(24008064)
BSc Computer Science

Project ID: 370

Module: SE3IP11

Supervisor: Professor Atta Badii

Submitted 20th April 2018

Abstract

ZEUS is a simulator that can be used to simulate a theoretical zombie infestation. The simulator has a dual use as it can be used to model the aforementioned zombie infestation, or the simulation can be used to model a realistic spread of a pathogen. Users can set up a simulation which contain the parameters of the zombie infestation (or pathogen) as well as the scenarios which may be involved. The simulation may also save these simulations as well as load and use the simulations, to allow for the simulations to be passed across different users and for later use.

Acknowledgements

I would like to thank Professor Atta Badii and Tom Thorne for supervising the project through its design and development as well as the documentation of the project.

Another individual that needs acknowledgement is Omar Cornut [1] for the development of the dear imgui (ImGUI) header library used extensively during the development of the ZEUS System.

Multiple collaborators that have contributed to the development (and maintenance) of the Simple Direct-Media Layer (SDL) [14] graphics library also need to be acknowledged for an indirect contribution to this project.

(Use of ImGUI under the MIT License)

(Use of SDL under the zlib and LGPL license)

CONTENTS

1. Introduction

Simulations are useful in emulating real-world scenarios in isolated conditions. The use of a simulation is advantageous as the effects of simulation is either limited or have no real / lasting effects; the possible effects of a scenario the simulation emulates can be observed without having to deal with the consequences. As such, a simulation would be very useful in modelling how a theoretical disease that causes people to become zombies would spread across the world / specific areas.

This project involves research from multiple fields of study, ranging from Computer Science (software development) to Epidemiology, to ensure that the simulator is relatively realistic but at the same time is not too complex that special training is required to use the simulator.

The main objective of this project is to produce a zombie simulator that runs a moderately realistic simulation of a scenario based on user inputs. The simulator must also be usable without the need of training to use the system i.e. the software must be easy to use, following the already set guidelines for user interface design.

A secondary objective is for the system to allow for customisation so that users are able to create scenarios other than the default scenario; therefore making the system be usable for simulations that aren't just the entire world (allows for more local simulations).

This document covers the development of the ZEUS System, beginning with the problem articulation, defining the problem that needs to be solved and defining the features the simulator should have to give purpose to the creation of the simulator.

After the problem articulation, research will be done to see existing simulators relevant to the project to see what features of each may be beneficial to include in the ZEUS System and more importantly how to improve upon existing simulators.

The main section of this document details the development of the ZEUS System and therefore the next few sections cover the design, implementation and testing of the prototype ZEUS System. These sections give an insight into the different processes of software development, as well as the different functions, classes and other components that are involved in the ZEUS System that allow it to perform as intended.

The final sections of the document will look at the ZEUS prototype and evaluate whether or not the problem stated beforehand has been solved and to evaluate ZEUS' overall performance in meeting its requirements. The future of the project will also be discussed to see whether improvements can be made as well as to review the overall organisation of the project.

This project is available in GitHub as a repository (as of April 2018). The repository includes all the source code and documentation involved in the project, for ease of marking as well as to provide access to any interested individuals who want to use the ZEUS System [2].

2. Problem Articulation

The main problem the project addresses is that there is a lack of simulations regarding the spread of zombies. There are simulators that perform tasks almost similar to the target objective (such as spreading a disease around the world); however other simulators do not also simulate the spread of zombies.

Other existing simulators also have a wide range of complexities, at a cost of simulation accuracy i.e. the more complex the prompts for input are, the more accurate the simulations; which leads to the secondary problem, which is that the ZEUS System must have a balance where the simulations are semi-realistic without the use of the simulations being too complex.

2.1 **Stakeholders**

2.1.1 Developer – Czar Ian Echavez

The sole developer involved with creating the solution for the problem outlined in Section 2. The developer is responsible for managing the parts of the project and ensuring that the deliverables (i.e. both the project development as well as the project documentation) are completed before the specified deadlines outlined by the University of Reading.

2.1.2 Project Supervisor – Atta Badii

The project supervisor keeps track of the development of the project, providing help when required by the developer as well as discussing progress made to the project and any possible improvements that can be done to the current state of the project.

2.1.3 End users

Any individual (or groups) interested in running zombie-based simulations would be possible end users of the simulator. The simulator can be used for running models for research into ways that the disease can spread around the (default) world, otherwise, more advanced users are able to create their own scenarios to run simulations that suits their own needs.

The simulator can also be used by individuals who are only interested in the subject of epidemiology; to further their interest as to how the study of epidemiology has real world applications.

3. Requirements and Analysis (Literature Review)

3.1 Case Studies (Existing examples)

3.1.1 GLEAM Simulator

The GLEAM (Global Epidemic and Mobility Model) Simulator [3] is a two-part system composed of a Client and server application. The system uses the server to run the simulations, none of the computations are done on the user's computer. The GLEAM system uses the client application to interact with the server. The client application itself is also split into smaller modules. The client application is split into a simulation builder, simulation manager and the simulation visualiser.

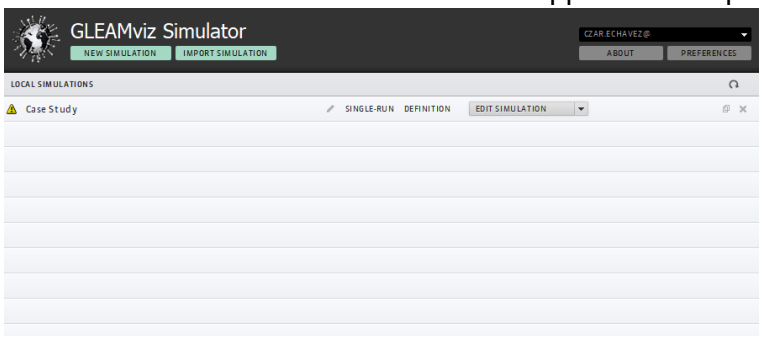


Figure 1 Simulation Manager

A user can track his/her simulations using the simulation manager. The simulation manager holds the simulations that are both complete and incomplete. Complete simulations can be submitted to the GLEAM server for the simulation to be run. Figure 1 shows the simulation manager's user interface.

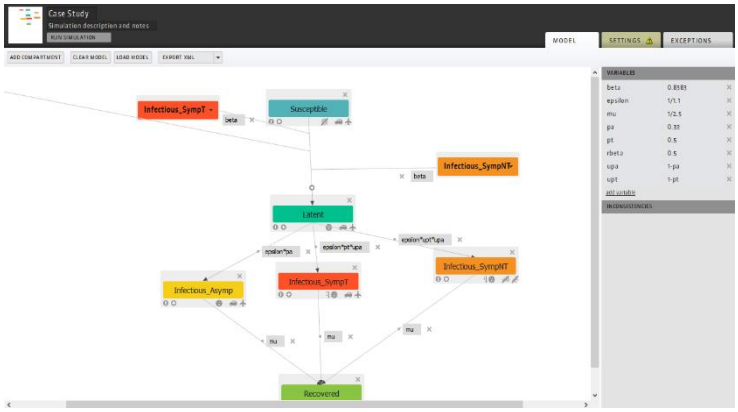
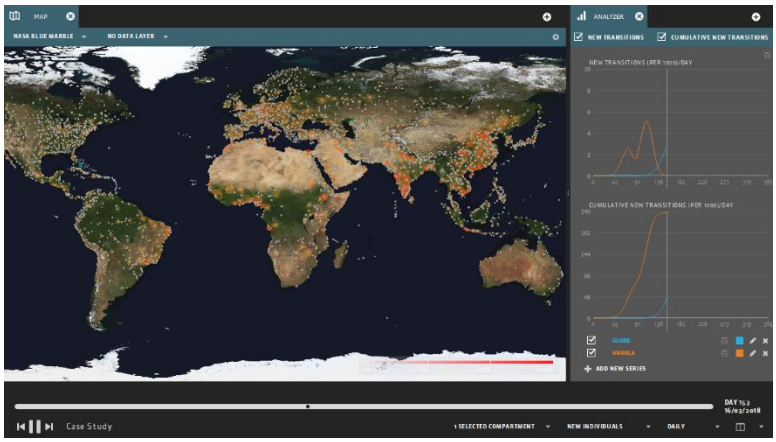


Figure 2 Simulation Builder

The simulation builder is where a user can create the spreading logic of the simulation i.e. how the disease propagates during the simulation as well as defining the starting point of the simulation. Figure 2 shows the user interface for the simulator builder.

The simulation builder works on the basis that each object on screen is a compartment and that connections between the compartments are transitions, each with a variable name (with set values) which are used to calculate the spread of the disease.



The simulation visualiser allows the user to see the results of the simulation. The visualiser has multiple settings and widgets to show the spread of disease in multiple ways. The world map remains the same however can be reskinned to highlight certain parts such as visualising which country is most susceptible to a pandemic. As shown in Figure 3, the visualiser also displays a graph of the new infections per day; other graphs can also be displayed.

Figure 3 Simulation Visualiser

3.1.1.1 Evaluation of GLEAM

GLEAM has a great user interface design; however, some aspects of the software require the user to refer to the manual [4]. The simulator builder has on screen help, however is not clear enough for a user to be able to make sense of what should be done in order for a simulation to be deemed runnable. Only after referring to the GLEAM user manual is it possible to understand how to create a simulation that the system deemed complete to be run. GLEAM breaking down its system into multiple pieces (i.e. the manager, builder and visualiser) also means that the software is not cluttered with a large amount of settings, ensuring that only the settings available in the current application is relevant to the action that the user needs to do; the simulation manager only allows the user to create and manage existing simulations, the builder only allows the user to create the disease mechanism and the simulation scenario, etc.

GLEAM is a simulator for realistic diseases, however, the goal of this project is also to create a zombie simulator. ZEUS still requires the use of disease spread, as it is assumed that the process of being turned into a zombie is that a person is first infected before becoming a zombie (assuming that the time for an infected to become a zombie is not the same as the incubation period)

Taking note of GLEAM's flaws would help in ensuring that the development of ZEUS has an easily understandable user interface as well as provide a good user experience when using ZEUS. ZEUS needs to be set to only one simulation logic to simplify the simulator, therefore this is taken into account during the development process.

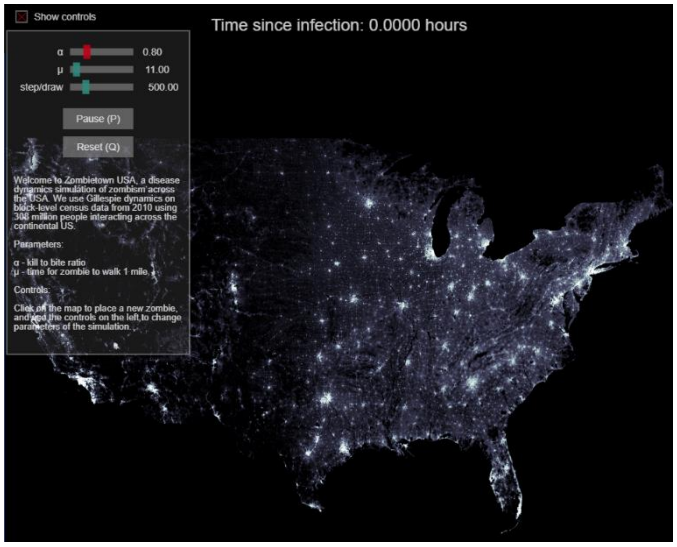


Figure 4 Zombietown USA demo

3.1.2 Zombietown USA

Zombietown USA [5][6] is JavaScript based application created by Matt Bierbaum and Alex Alemi.

The application is a simple simulation of how the zombies propagate through a population; the areas of high population are the lighter parts of the map, shown in Figure 4. The application only simulates the spread of zombies within the US and does not spread around the world.

The user interface for the application is also very simple, requiring only 4 different inputs from a user; the 3 variables the user can set, and the location(s) where the zombies can spread from.

Figure 5 Infection Spread

Figure 5 shows the simulation after a few steps to show the spread of the zombies. The zombies are shown in red while healthy populations are still white (and grey). The simulator emphasises that the higher the population, the faster the zombies propagate through the region, hence the spread of zombies showing bumps on regions of high populations.

Figure 6 Simulator Variables

Figure 6 shows the 3 variables that the user is able to change. The sliders at the top of the interface denote the 3 variables that affect the simulation; these variables affect how the disease propagates (the chance of an infected individual turning into a zombie), how fast the disease travels (and therefore how fast a zombie should travel) and how many simulation cycles should be done before drawing the result to the screen.

The user is able to set the starting point of the simulation by clicking on the map to "place" the initial zombie or if multiple points are clicked, zombies.

3.1.2.1 Evaluation of Zombietown USA

The user interface for this application is both easy to understand and use. The simulation is largely automated and begins as soon as the user clicks on the map where the simulation should begin from. The parameters to control the simulation is also very easy to understand and manipulate; the explanation of how the variable affects the simulation is shown where the user changes the parameters so that the user is able to know what should happen before they choose to change the variables.

The simulator's simplicity however, causes it to be non-realistic. The simulator is too simple and cannot simulate the fact that the disease is able to spread to other cities instead of steadily moving from the starting point (a user can simulate this by having multiple start point but should be automatic). The simulator also only simulates the spread of zombies; this can be improved by also showing the spread of the disease, however this scenario may be where an infected individual becomes a zombie straight away.

The simple UI design should be taken into account in the development of ZEUS; a simple UI design makes the simulator easy to use and understand, and if a component needs explanation, there must be a way for the user to understand the item without needing to refer to a user manual.

3.2 **Simulation Mechanisms**

3.2.1 Mathematical model of a zombie disease

Knowing how a zombie disease will spread across a population is required by ZEUS in order to perform the calculations for the simulator and therefore making a more realistic simulation. Making a mathematical model of the simulations also means that the simulations remain consistent across the different copies of the simulator (excepting the fact that RNG may cause slight differences).

A collaborative study by statisticians from the University of Ottawa and Carleton University [7] created a mathematical model of a spread of zombies across a population. The model breaks down the model into 3 distinct classes: Susceptible (people that can be infected), Zombies, and Removed (which are removed from the population pool if a zombie is destroyed or an uninfected individual dies).

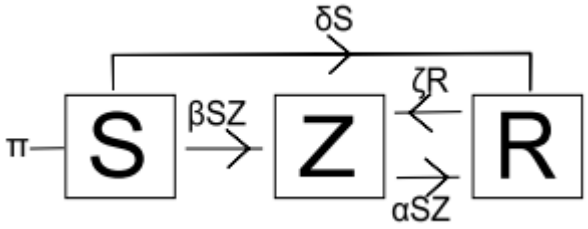


Figure 7 Basic infection model

Figure 7 (taken from [7]) shows the basic version of the model; S, Z and R representing Susceptible, Zombie and Removed respectively.

π is an input into the population pool, the model assumes a birth rate i.e. more humans being born.

The model shows susceptible individuals becoming a zombie but cannot become uninfected, however a "removed" individual is able to be revived as a zombie again. The transitions have parameters which define the rate at which an object becomes another object e.g. βSZ is the rate of infection, αSZ is the rate of zombie elimination, ζR is the rate of reanimation (of zombies), etc.

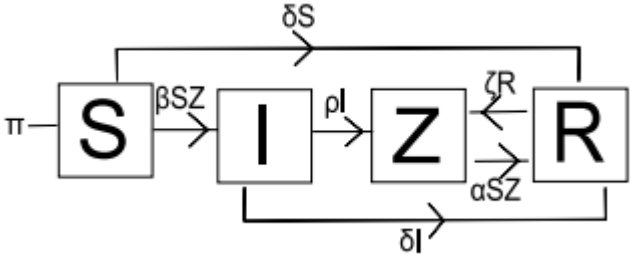


Figure 8 SIZR Model (with latent infection)

Figure 8 (taken from [7]) takes the basic model further by implementing an intermediate class between Susceptible and Zombies. This class is for infected individuals – people who have been bitten but have not yet become zombies.

In this model, new transitions are added; the infected become zombies at a rate ρI and infected individuals that do not transition into zombies at a rate of δI

The model in Figure 8 can be easily adapted to serve as the disease spread logic for ZEUS. Some alterations are done in order to simplify the model as well as ensure that the simulation runs for a limited length of time (i.e. finishes when either there are no more

infected and/or zombies); the use of π introduces more individuals into the simulation and would essentially cause the simulation to last longer than required and possibly come to an equilibrium state where the disease infects at the same rate as the birth rate.

4. Design Specification

4.1 ZEUS System model

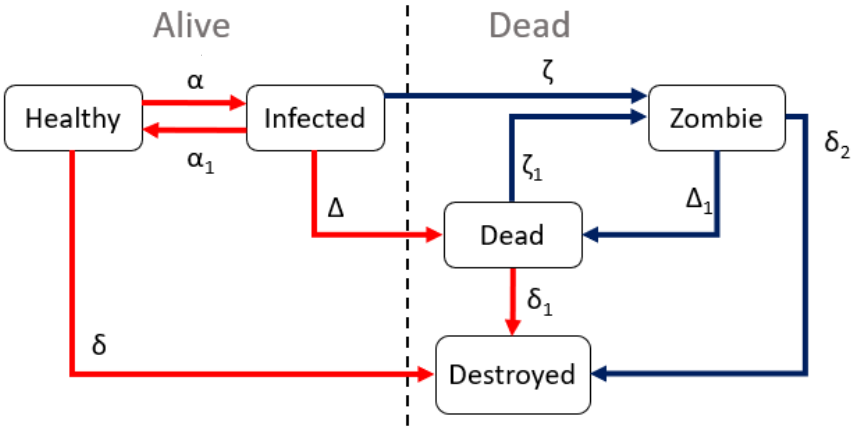


Figure 9 Proposed ZEUS Model

Figure 9 is the proposed simulation model for ZEUS; this model is based from the SIZR model discussed in 3.2.1 Mathematical model of a zombie disease.

The main alteration to the SIZR model is the removal of the birth rate, effectively making the model in Figure 9 a closed system i.e. the number of population in the simulation is set and no new individuals are introduced into the simulation.

Another alteration to the SIZR model is the splitting of the “Removed” class into a “Dead” and “Destroyed” class. Both act as “Removed” as in the SIZR model, however dead classes can be reanimated into zombies, and destroyed is the end point for the model i.e. once an object is deemed destroyed, it is considered fully removed from the simulation.

Figure 9 also shows the grouping of all the classes; the classes are split into “Alive” and “Dead” (not to be confused with the dead class). This clearly shows the logic behind the transitions, as dead classes cannot come back to life as either infected or healthy. The model also considers zombies as dead, therefore even if the cure is found, the zombies persist until they are destroyed.

As for the possibility of cures, an “Immune” class was planned to be implemented, however, this is scrapped, as once the cure is triggered, the healthy class automatically is converted into the immune. The infectivity α will be set to 0, meaning no more infections can occur, as well as a higher rate of infected of being cured and therefore becoming immune.

The ZEUS model also shows 2 different forms of transitions; Figure 9 shows 2 different coloured transitions in the model. Red coloured transitions are for when the simulation is running only to show the spread of the disease without the zombies. In a non-zombie based scenario, dead and destroyed classes are effectively the same. In a zombie based scenario, the blue coloured transitions would be used.

4.1.1 Breakdown of each variable

Figure 9 has multiple variables attached to each transition states; these are variables that the simulation uses to calculate the portion of the population that undergoes the transition e.g. how many healthy individuals are to become infected, therefore the same amount added to the infected pool is the same amount the healthy pool is reduced by.

These variables are further broken down by different substituent parameters. The variables are affected by a singular country's parameter such as a country's climate; this allows the simulation to show how different the infection propagates through different countries.

α	Infection rate	This variable affects the rate of infection of the disease. The higher this value, the more healthy individuals transition into an infected state. A country's climate can affect the overall value of the infection rate. Multiple other parameters set by a user can affect this variable; Transmission vectors such as air, water or zoonotic (animals) can increase α. Simulation settings (such as allowing to simulate air travel and sea travel) also has an (indirect) affect on the infection rate
α_1	Recovery rate	The recovery rate affects how much of a population recovers from being infected. Normally if infected individuals recover, they become immune to the disease, this is not simulated; therefore the recovered individuals are returned to the healthy pool (unless a cure is found, in which case the healthy pool is converted into the immune pool. This variable is affected by a country's research budget (the research budget emulates the country's funding of their health care system, the default data included is taken from the CIA World Factbook Health Expenditure for each country [8])
δ	"Natural" death rate	The natural death rate is the rate at which healthy individuals die naturally i.e. without the effects of the disease. This emulates the fact that individuals can die of old age (natural causes) and/or accidents (and possibly other diseases not currently being the focus of the simulation). These individuals are assumed to not be infected by the disease and therefore go straight into the destroyed class instead of the dead class, as the dead class should only have individuals who were infected by the disease; this also prevents uninfected dead people from being reanimated into zombies when they should not be. The natural death rate is affected by the research budget of the country (for the same reasons as in the recovery rate) and by the GDP; higher GDP countries are assumed to have a higher quality of life (such as introductions of health and safety, food safety laws, etc)

		The decay rate is the rate at which the dead class slowly decays i.e. the corpse rotting. The dead class turns into the destroyed class, effectively removing them from the simulation; this is because as the corpse decays it is no longer possible for the object to be reanimated into a zombie.
δ_1	Decay rate	The decay rate is affected by the country's climate; a warmer climate causes decay of corpses to be much faster, therefore the transition from dead to destroyed occurs much faster in warmer countries. The opposite happens for colder climates, colder temperatures slow down the rate of decay and therefore decay rate decreases.
δ_2	Zombie elimination rate (permanent)	The zombie elimination rate is the rate at which zombies are **permanently** removed from the simulation, whether it be zombies decaying naturally or by being destroyed by alive humans. Since the zombies can decay, the variable is affected by the climate of the country, similar to the decay rate. The country's military budget also affects the zombie elimination rate, as a better funded military is assumed to be more effective in removing zombies within their country.
Δ	Death rate from disease	Death rate from disease is the variable that the simulation uses to determine how many of each infected die from the disease. In Figure 9 the infected are not classed as "destroyed" when undergoing this transition; instead the transition is towards the dead class, which means that the infected individual that dies can be reanimated into a zombie. The death rate is affected by the research budget of each country, similar to whether or not the country is effective in being able to keep the person alive (as in the recovery rate).
Δ_1	Zombie elimination rate (non-permanent)	Similar to δ_2 above, this variable acts the same way, removing zombies from the zombie population pool; however instead of the zombies being permanently removed, they are instead converted into the dead state, which means that the zombies can still reanimate. This variable is also affected by the military budget of a country, for similar reasons as δ_2.
ζ	Zombie conversion rate	The zombie conversion rate is the variable used to determine how much of the infected are converted into zombies instead of dying. This variable is not affected by any scenario parameters (country values), instead this is a variable that can be manipulated in the simulation settings.
ζ_1	Reanimation rate	The reanimation rate is the variable used to determine how often the dead classes should be reanimated into zombies. Similar to ζ, this variable is not affected by anything and is a variable that can be manipulated in the simulation settings

4.2 **Use Cases**

Element: ZEUS System GUI	**Use Case ID:** 1

Stakeholders/Interested Parties:
- Epidemiologists
- Pathogen Researchers
- Hobbyists

Primary Actor: Software User

Description:
The user of the software will need a way to interact with the software. A Graphical User Interface will provide a user with proper prompts as well as output the correct responses depending on the input(s).

Trigger(s):
- User gives mouse inputs via clicking or moving move around the screen
- User gives key inputs via the keyboard or on screen keyboard

Conditions:
- User needs to have a keyboard
- User needs to have a mouse
- Keyboard and mouse need a way to communicate with the computer running the software, either via physical wire or wireless connection

Event flow:
- User gives input into keyboard or mouse
- Keyboard or mouse transforms input into string/numerical values the computer can process
- The input is passed into the software
- The software check if the mouse or keyboard input is relevant and performs actions based on the input

Alternate flow:
- User inputs values to keyboard and mouse but neither are connected to the computer, therefore the input is not detected
- User inputs values but the software is not running therefore inputs are not detected
- User inputs values but the inputs are irrelevant to the software

Element: ZEUS Scenario Creator	**Use Case ID:** 2

Stakeholders/Interested Parties:
- ZEUS System users

Primary Actor: Advanced ZEUS Users

Description:
The Scenario creator is used to create new scenarios, maps which the main simulator system can use in order to run simulations on.

Trigger(s):
- User gives mouse inputs via clicking or moving move around the screen
- User gives key inputs via the keyboard or on screen keyboard
- User uses the GUI elements to interface with the system

Conditions:
- User needs to have a keyboard
- User needs to have a mouse
- Keyboard and mouse need a way to communicate with the computer running the software, either via physical wire or wireless connection

Event flow:
- User gives input parameters to create a new scenario; the scenario map (image) to be used and the name of the scenario
- The user populates the scenario map with countries; each country has its own values such as population, GDP, research budget, military budget and climate
- User connect countries together by adding “links” between them
- User finished creation of simulation by saving it to a file location of their choosing

Alternate flow:
- User opens an already existing scenario
- User tells the program which file is to be opened for editing
- The user makes necessary edits to the scenario data
- User saves the file once finished

Element: ZEUS System	**Use Case ID:** 2

Stakeholders/Interested Parties:
- ZEUS System users

Primary Actor: ZEUS Users

Description:
The main part of the system which runs the simulations and uses the scenarios created by the scenario creator

Trigger(s):
- User gives mouse inputs via clicking or moving move around the screen
- User gives key inputs via the keyboard or on screen keyboard
- User uses the GUI elements to interface with the system

Conditions:
- User needs to have a keyboard
- User needs to have a mouse
- Keyboard and mouse need a way to communicate with the computer running the software, either via physical wire or wireless connection

Event flow:
- User gives input parameters to create a new simulation.
- User defines which scenario should be used in the simulation
- User defines the simulation parameters
- User can save the simulation setup and then run the simulation

Alternate flow:
- User opens an already existing simulation
- User tells the program which file is to be opened for editing
- The user makes necessary edits to the simulation data
- User saves the file once finished

4.3 **Menu Bar options**

The menu bar is the main place where a user is able to access the different windows that allow them to edit the scenario / simulation values. The design of the menu bar must be clear so that the options are placed under the correct menu headings.

4.3.1.1 ZEUS Scenario Creator

4.3.1.1.1 Menu bar hierarchy

Figure 10 Scenario Creator

Main menu bar hierarchy

The Scenario creator's GUI will consist mainly of the main menu bar and the windows that pop up when the user selects an option from the main menu bar. There are 4 main options that the user can choose from; file, view, help and quit.

The "File" option brings up commands that are relevant to dealing with the files the scenario creator is meant to handle. Figure 10 shows that the option has 4 sub menus that the user can pick.

- "New scenario" allows the user to create a new map (scenario) that a simulation can be run on.
- "Open scenario" allows the user to open a previously created scenario; This option allows users to continue editing a previously created scenario, ensuring that a user does not need to start all over again
- "Save" allows the user to save the files, so that the scenario parameters are usable as an input for the simulator as well as the user is able to continue editing the scenario for later use.
- "Save As" allows the user to save the scenario under a different file name, such as in the case where the user wants to keep the original file and save the current edits to the file

"View" has 2 options that are only relevant when a scenario is currently being worked on; the options should only be used when either a scenario is newly created or loaded, as the options are useless when the program does not have a loaded scenario. Figure 10 shows that view has 2 sub menus from which the user can pick from.

- "View all countries" is an option that allows the user to see all the countries involved in the scenario. The countries are to be shown in a pop up window which displays the country unique ID as well as the country name and options for what to do with the countries. These 2 options are whether or not to edit the country or to delete the country.
- "Add new country" is an option that allows the user to add a new country. The option brings up the window which allows a user to create a new country; this window can also be brought up by clicking on the screen to select a colour region as a country.

"Help" is an option that will provide more information to the user about how to use the simulator. There are 2 options, one is a local copy of the manual and another is an online copy. This is for further explanation of how the simulator works; should the user not understand what to do even with the on screen explanations.

"Quit" only has 1 option. This option is the menu option that closes the application; this is done to ensure that the user does not accidentally close the application when the user was not meant to i.e. if a user misses the "help" option and presses the "quit" option, having the sub menu "Close application" prevents the user from closing the application accidentally.

- File
  - New Simulation
  - Open Simulation
  - Save
  - Save As
- Edit
  - Edit Simulation
  - Change Scenario
- View
  - Scenario details
  - Simulation details
  - View all countries
- Simulation
  - Run
  - Pause
  - Reset
- Preferences
  - Appearance preferences
- Help
  - Online manual
  - Local manual
- Quit
  - Close Application

Figure 11 ZEUS Main System

Main menu bar hierarchy

4.3.1.2 ZEUS Main System

There are 2 main UI elements used in the ZEUS Main System; the main menu bar similar to the Scenario creator, and the information box on the right side of the application window. The main menu bar has options which allows a user to edit the simulation parameters as well as control the visualisation of the simulation.

The main system has more options than the scenario creator as the main system handles more items than the scenario creator.

The “File” option has sub options which allow the user to handle simulation data. Figure 11 shows the 4 sub options users can pick.

- “New simulation” allows a user to create new simulations
- “Open Simulation” allows a user to load a previously created simulation. This is so that a user does not have to start from scratch when running the same simulation in another session
- “Save” and “Save As” allows the user to save the current edits to the simulation currently being worked on.

The “Edit” option has 2 sub options. The 2 options are relevant to the variables that will be used during the running of the simulation.

- “Edit Simulation” is where the user is able to change the simulation parameters, such as infection rate, whether to run a zombie simulation, etc.
- “Change Scenario” is where the user can change the scenario that the simulation will run on. The simulator engine is able to use the same simulation values on different scenarios.

The “View” option is slightly different in the simulator than it is in the scenario creator. The view option is to allow the user to get a more detailed view of the objects involved in the simulation to be (or currently being) run.

- “Scenario details” opens a window which lists the global scenario data i.e. total population, number of countries, etc, involved in the scenario.
- “Simulation details” opens a window which shows the simulation information
- “View all countries” lists all the countries involved in the simulation. This is also where a user chooses the initial country (or countries) that the disease or zombies will originate from

The “Simulation” option deals with whether or not the simulation should run, be paused or reset. There are shortcuts to run and pause the simulation (by pressing space bar), but the reset option must be chosen to reset the simulation values.

The “Preferences” option only has one sub option; this option opens a window which allows the user to change the background colour of the map, to allow for the simulation to be easier to see if the scenario map happens to be in the same hue as the background.

“Help” and “Quit” options do similar things in the simulator as it does in the scenario creator. Both “Help” options lead to the manual and the “Quit” option closes the application.

5. Development Tools

Different tools are used in the development of the ZEUS System; these tools are used to ensure the development of ZEUS goes smoothly and that any mistakes made during the development can be easily undone to prevent a long delay. The tools assist in automation of certain aspects of software development which would otherwise need to be manually done by a software developer – an example of such automation is the use of a build automation tool; a developer may need to manually create a makefile to compile the source code, however with the use of a build automation tool, the compilation of the source code can be done without the developer making the makefile.

5.1 Visual Studio IDE

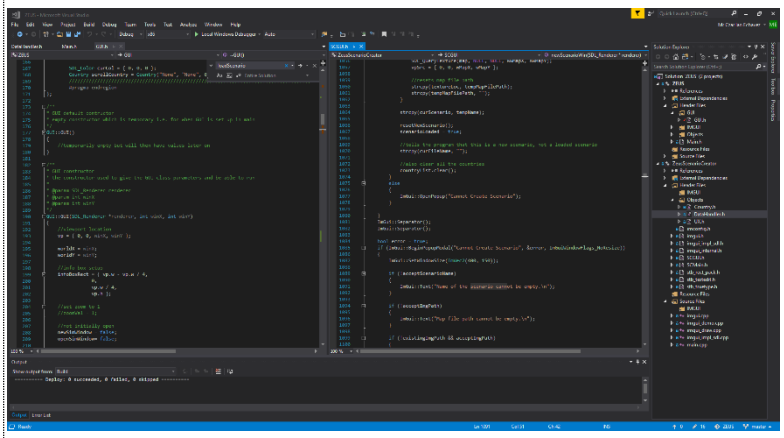


Figure 12 Visual Studio Layout while developing ZEUS

Visual Studio (Community) 2017 [9] is an integrated development environment provided by Microsoft. There are 3 editions of the IDE; Professional, Enterprise and Community.

There are a range of differences between the different editions of Visual Studio; Professional is the complete package, which involves extra items that is not limited to the software development tools – these would be training for the use of the tools themselves. Enterprise edition has tools relevant to collaborative development i.e. tools that allow a business to have multiple developers working on the same project. Other tools required for testing are also included in the Enterprise edition.

The Community version is a basic and free edition of Visual Studio. The edition has similar functionality as the Professional edition however, the software developed using the IDE is restricted to individual developers and that the software being developed are to remain open source.

Community edition is chosen to be used for developing the ZEUS System, as the task will be undertaken by one individual as well as the Professional and Enterprise editions being too costly [10]. The software being developed is planned to remain open source afterwards, therefore the Community edition is the optimal choice.

5.2 Git, GitHub and GitHub Desktop

Figure 13 Git Project Logo

Git [11] is an open source version control system that keeps track of changes in computer files and helps coordinate the files among multiple users / computer machines. Git is primarily used for the management source code during software development however, can be extended for use with other files.

Git was created in 2005 by Linus Torvalds and used to help with the development of the Linux kernel; the git software is open source and free for use under a GNU General Public Licence (GPL).

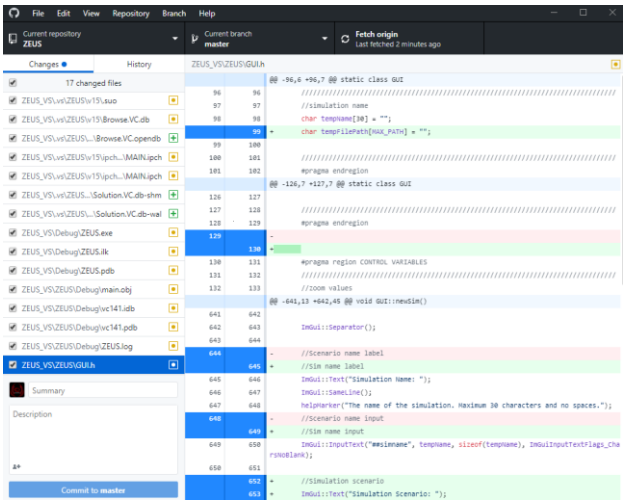


Figure 14 GitHub Desktop UI

GitHub is web-based hosting service for version control based on Git. GitHub also provides additional functionality than the source code management provided by Git; GitHub also has bug tracking, public requests and project wikis for each individual repository it hosts. Development of ZEUS heavily involves the use of version control provided by GitHub; the repository for the project is public and available for viewing (and release versions are available) [2].

GitHub Desktop is a desktop client that is used to interface with the online repository stored in GitHub; there are other clients that can be used to do this, however GitHub desktop is used in the development of ZEUS as it is the most widely available and recommended by GitHub.

5.3 **Notepad++**

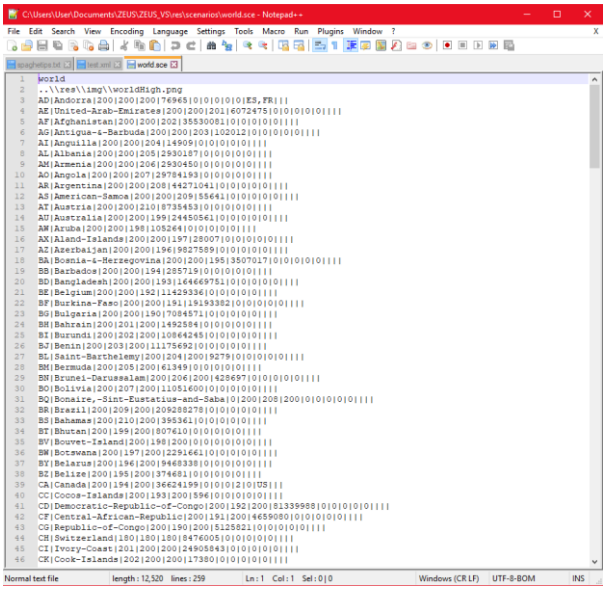


Figure 15 Notepad++ and a scenario file

Notepad++ [12] is an open source text editor started by Don Ho. Notepad++ has support for programming languages as well as provides editing text of any “text” files. Notepad++ has better utilities than the stock notepad provided by the Windows OS as well as the Vim text editor on Linux platforms.

Notepad++ is mainly used for editing scenario and simulator files used by the ZEUS System and is not used to edit the program’s source code. The UI for Notepad++ is much better than the default text editor on the Windows OS, as well as Notepad++ having a better find and replace function, which is helpful in editing the scenario files.

5.4 **External dependencies / libraries**

The creation of the ZEUS System requires more than just the use of the standard C++ library. C++ itself does not have a standard graphics library (as of the writing of this report, there had been proposals to integrate the “cairo” library [13]); this also means that since C++ does not use a graphics library, it does not have a GUI library. ZEUS’s development requires the use of libraries that allow the rendering of graphics.

Two main external libraries involved in the development of the ZEUS System are SDL and ImGui. SDL is an open source library and ImGUI is a header library being developed by Omar Cornut.

5.4.1 SDL (Simple Direct-Media Layer)

Figure 16 SDL Project logo

SDL [14] is a cross-platform library originally created by Sam Lantinga [15] which was first released in 1998. The current version is SDL2 (2.0.5 used throughout development), which is worked on by multiple collaborators. The library is cross platform and can be used on different programming languages and operating systems. SDL also includes extensions that allow for managing video, audio, hardware and networking.

5.4.1.1 Licensing

SDL is also included with the zlib license, which allows users to use the provided files "as-is" or if modified, the original authors must be properly represented. SDL files will not be edited throughout the development of the ZEUS System and therefore this license is not violated.

Along with a zlib license, SDL has a GLPL (GNU Lesser General Public License) attached to its use; GLPL means that if software uses a GLPL covered library, then it should allow user(s) to be able to be link with a newer version of the LGPL covered program (usually a dynamic linked library file that can be updated <this is extremely simplified in this sentence, full GLPL explanation is in [16]>)

5.4.1.2 Use in ZEUS System

The ZEUS System uses the SDL library to render the graphics (especially the scenario map) that the simulation uses. The SDL library is also the basis to which the application window is created; the window is not created using the Win32 API and is instead created via the SDL library. SDL is also used to take in user inputs such as key presses and move movements in order to make the correct outputs to a user's input.

5.4.2 ImGUI (Immediate mode GUI)

ImGui also known as dear imgui [17] is a GUI header library created by Omar Cornut [1]. The library is a collection of C++ header and CPP source files which need to be included in the program source; the source files needs to be compiled with the source code and acts as part of the source code instead of being linked.

The main ImGUI code is based solely for C++, however, multiple contributors have created API wrappers for other languages – allowing for ImGUI to be used for other programming languages. The library is also usable with other frameworks, such as with SDL in the case of the development of ZEUS.

5.4.2.1 Licensing

ImGUI is licensed using the MIT License, allowing for ImGUI to be used in the ZEUS System with some modifications so long as the original creator is given sufficiently credited i.e. their name and the copyright notice is included in the software. The modifications done to the ImGUI files are mostly for configuration of the ImGUI components as well as the integration of the files to the ZEUS System.

5.4.2.2 Use in ZEUS System

The ZEUS System utilises ImGUI as the header that handles the look of the system's GUI for both the main system and the scenario creator. Most of the GUI components such as the main menu bar, side bar (for the main system) and system windows are handled by the ImGUI library. As stated previously, the library is used in conjunction with the SDL library to control what the user sees on screen; SDL provides the window and the rendering framework which ImGUI utilises to properly render what should be shown on screen according to the user's input

6. System Development

Just as extra context, the setup for both the scenario creator and the simulator are similar up until Section 6.2 and Section 6.3, however, the classes specific to each piece of software are differentiated by the fact that anything linked to the scenario creator is preceded by “SC”.

6.1 Window and Renderer

The development of both the scenario creator and the simulator begin with the creation of the window and renderer. This is to be able to visualise the result of later development on the window; effectively making the overall development of the ZEUS System via a top-down approach.

Both the scenario creator and the simulator use the same method to create their window, with minor variations to allow for the different tasks each software is made for. This will be detailed in the coming sections.

6.1.1 SDL variables

```
53  private:
54      //constant values
55      const int ZOOM_IN = 1;
56      const int ZOOM_OUT = 0;
57
58      //window size
59      int winWidth, winHeight;
60
61      //simulation window
62      SDL_Window *window;
63      int winX, winY;
64      //simulation renderer
65      SDL_Renderer *renderer;
66
67      //bool to store application run
68      bool appRun;
69
70      //SDL Events
71      SDL_Event eventMain;
72
73      //mouse position
74      SDL_Point mousePos;
75
76      //opengl context
77      SDL_GLContext glcontext;
78
79      //time handling
80      int frames = 0;
81      float fps = 0;
82      int fpsLimit = 120;
83
84      //panning
85      bool isPanning = false;
86
87      //the GUI class used with the Main
88      SCGUI gui;
```

Figure 17 SCMain variables

Figure 17 shows the SDL variables used in the SCMain.h object; this may vary between the Scenario Creator and the Main System, however both use the same method to make a window. These variables determine the window size as well as whether or not the program is still running (the boolean appRun).

There are also a few SDL variables that are used in conjunction with SDL to ensure that the SDL component of the software acts as it should.

SDL Window and SDL Renderer are both used to render the graphics used by the program. The SDL Window provides the program window where all the graphics of the program are rendered (by the SDL Renderer).

SDL Event is the component of SDL which is used to take in user events (inputs such as mouse movements, key presses, etc). This variable is evaluated in a switch case, where the program executes code according to the state of the SDL Event variable.

Both software have the ability to pan and zoom into the scenario map; with a little difference in the main simulator software as the zooming and panning is only done to a section of the window as opposed to the whole map being displayed to the full window in the scenario creator.

6.1.1.1 SDL Window

```
// Setup window
SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER, 1);
SDL_GL_SetAttribute(SDL_GL_DEPTH_SIZE, 24);
SDL_GL_SetAttribute(SDL_GL_STENCIL_SIZE, 8);
SDL_GL_SetAttribute(SDL_GL_CONTEXT_MAJOR_VERSION, 2);
SDL_GL_SetAttribute(SDL_GL_CONTEXT_MINOR_VERSION, 2);

//create the window
window = SDL_CreateWindow(
    "ZEUS Scenario Creator",
    SDL_WINDOWPOS_CENTERED,
    SDL_WINDOWPOS_CENTERED,
    //window width
    winX,
    //window height
    winY,
    //show window on screen
    SDL_WINDOW_SHOWN
    //resizable window
    | SDL_WINDOW_RESIZABLE
    //maximised window
    | SDL_WINDOW_MAXIMIZED
    //opengl window
    | SDL_WINDOW_OPENGL
);

//get display size
//debug
//SDL_GetWindowSize(window, &winX, &winY);

//opengl context
glcontext = SDL_GL_CreateContext(window);
```

Figure 18 Creating a window

Figure 18 shows the process of creating an SDL Window. The 5 lines above the actual creation of the window is to set up the SDL GL Attributes in order to allow the integration of ImGUI with the SDL Window. ImGUI uses OpenGL to render items, similar to how SDL renders to the screen, however both their settings have to be set to allow proper rendering.

After these attributes are set up, the creation of the window is done; normally this can all be done in one line, however for clarity of code, each parameter is placed on a new line.

The function SDL_CreateWindow creates the window, taking in multiple parameters; The first parameter is for the name of the window, in this case “ZEUS Scenario Creator”.

The next parameters are the position of the window when opened; in the X and Y axis. After this are the window’s width and height (winX and winY respectively).

The final parameter(s) are the different flags for the window; these flags determine the window’s behaviour.

In the final parameter, each flag is separated by a pipe (“|”); these pipes are bitwise OR operators; SDL stores the window flags in a byte, each of the flags have a specific integer value, when the OR bitwise operation is done to all the flags, the bytes are combined, allowing the function to distinguish which flags are to be used.

6.1.1.1.1 SDL Window error checking

```
//check if window was created
if (window == NULL)
{
    std::cout << "SDL Window error: " << SDL_GetError() << std::endl;
    //exit the program
    exit(0);
}
```

Figure 19 Error checking in SDL Window creation

Once the window is made, it is best to check if it has been properly created. This is done by checking if the window is null. If the window is null, the program is closed; if opened via the console, the program prints the error, allowing an advanced user to determine the cause of the error.

```
//continue setup
else
{
    //initialise IMGUI
    ImGui_ImplSdlGL2_CreateDeviceObjects();
    ImGui_ImplSdlGL2_InvalidateDeviceObjects();
    ImGui_ImplSdlGL2_Init(window);

    //find GPU drivers
    int oglIdx = -1;
    int nRD = SDL_GetNumRenderDrivers();
    for (int i = 0; i<nRD; i++)
    {
        SDL_RendererInfo info;
        if (!SDL_GetRenderDriverInfo(i, &info))
        {
            if (!strcmp(info.name, "opengl"))
            {
                oglIdx = i;
            }
        }
    }

    //create the renderer, linking the number of GPU drivers
    renderer = SDL_CreateRenderer(window, oglIdx,
        SDL_RENDERER_ACCELERATED
        | SDL_RENDERER_PRESENTVSYNC
    );



    //initiate glew
    glewExperimental = true;
    glewInit();

    //setup ImGui style
    //ImGuiStyle &style = ImGui::GetStyle();
    //style.Colors[ImGuiCol_MenuBarBg] = ImVec4(0.6f, 0.0f, 0.0f, 1.0f);

    //allow transparency
    SDL_SetRenderDrawBlendMode(renderer, SDL_BLENDMODE_BLEND);

    gui = SCGUI(renderer, winX, winY);
}
```

Figure 20 Creating the renderer

6.1.1.2 SDL Renderer

The SDL Renderer is created once the program checks the window for errors; if the creation of the SDL Window is successful, the program then creates the Renderer.

The renderer is always made after the window, as the creation of the renderer is dependent on the existence of a window; the renderer is attached to whichever window the renderer will render on.

This section is also where the ImGUI components are properly initialised; similar to the renderer, the ImGUI renderers are dependent on the existence of a window, which is the parameter passed in the initialisation of ImGUI.

The section after the initialisation of ImGUI deals with the checking of the number of render drivers which SDL can utilise. This means that hardware acceleration is possible; hardware acceleration allows for more efficient performance of functions than what is normally possible on a CPU i.e. rendering graphics are more effectively done via the GPU.

The SDL function SDL_CreateRenderer creates the renderer which is used in the program. The function takes multiple parameters as input to create a renderer relevant to its use in the system. The first parameter is the window which the renderer is to be attached to, in this case, the window that was created in section 6.1.1.1. The second parameter is the number of graphics drivers that the renderer is able to utilise; this will vary between machines.

Figure 21 Screen tear example

Source: https://en.wikipedia.org/wiki/Screen_tearing#/media/File:Tearing_(simulated).jpg

The final parameter(s) are the flags that determine the renderer behaviour. Similar to the creation of the window, this function also makes use of bitwise operations. The renderer is given an accelerated flag and a VSync flag; the accelerated flag allows the renderer to use hardware acceleration, utilising the graphics drivers whereas the VSync flag tells the renderer to wait until the whole image (frame) is rendered before being rendered to the screen.

VSync is a display process which ensures that no screen tear occurs i.e. prevents a partially completed (incomplete) frame from being rendered and therefore causing graphics errors as shown in Figure 21.

6.1.1.3 SDL Event

```
//check for inputs
while (SDL_PollEvent(&eventMain) != 0)
{
    //process event for IMGUI
    ImGui_ImplSdlGL2_ProcessEvent(&eventMain);

    //process events via SDL
    switch (eventMain.type){ ... }

    //ctrl + S shortcut
    if (key[SDL_SCANCODE_LCTRL] && key[SDL_SCANCODE_S] ||
        key[SDL_SCANCODE_RCTRL] && key[SDL_SCANCODE_S])
    {
        gui.ctrlS();
    }
}
```

Figure 22 Event Handling in Scenario Creator

The SDL Event is used to determine the keyboard/mouse inputs that the user does. The while loop in Figure 22 is executed only when there is an event that needs to be evaluated.

If there is an event, the loop is executed; the loop has a switch case that checks for the event type as well as an if statement that checks for a specific combination of key presses (in this case, control + S as a shortcut to save the scenario).

[There are more shortcuts, such as for new scenario and open scenario, Figure 22 only provides an example]

6.1.1.3.1 Switch case

```
case SDL_QUIT:
    appRun = false;
    break;
```

Figure 23 Event for when window is closed

6.1.1.3.1.1 Closing the window

When the window is closed, (by pressing the X button on the top right (for Windows)) it is assumed that the user no longer wishes to use the program, therefore, the value of appRun is set to false, which causes the program to end.

6.1.1.3.1.2 Mouse clicks

If the event is a mouse click, the button clicked should be determined, as the left click acts differently to the right click. The left click is to choose options, meanwhile the right click is a click and drag operation which is used to pan the screen.

[Figure 24 shows that isPanning is set to true, in a later section, the event checks if the right click is released, which sets isPanning to false, to prevent the user from dragging the map forever]

```
    //while clicking the left button
case SDL_MOUSEBUTTONDOWN:
    if (eventMain.button.button == SDL_BUTTON_LEFT)
    {
        gui.leftClick();
    }
    //middle mouse button
    if (eventMain.button.button == SDL_BUTTON_RIGHT)
    {
        isPanning = true;
    }
    break;
```

Figure 24 Left and right click events

```
case SDL_MOUSEWHEEL:
    //mouse wheel for zoom control
    if (eventMain.wheel.y == -1)
    {
        //zoom in for scroll up
        gui.zoom(ZOOM_IN);
    }
    else if (eventMain.wheel.y == 1)
    {
        //zoom out for scroll down
        gui.zoom(ZOOM_OUT);
    }
    break;
```

Figure 25 Mouse wheel case

6.1.1.3.1.3 Mouse wheel

Figure 25 shows the case to check if the mouse wheel has been scrolled up or down. In both the scenario creator and the simulator, a scroll up zooms in, making the viewport expand and concentrate on a certain position of the image, whereas a scroll down does the opposite.

The mouse wheels are the main events looked at relating to the zooming in and out of the scenario map, however are not the sole control inputs to zoom in and out

```
case SDL_KEYDOWN:
    if (eventMain.key.keysym.sym == SDLK_EQUALS)
    {
        gui.zoom(ZOOM_IN);
    }
    else if (eventMain.key.keysym.sym == SDLK_MINUS)
    {
        gui.zoom(ZOOM_OUT);
    }
    break;
}
```

Figure 26 Key Press case

6.1.1.3.1.4 Single key presses

Single key presses act somewhat different to the key presses shown in section 6.1.1.3. Single key presses do not execute any code when held down, instead requiring to be “refreshed” i.e. the key must be pressed once and execute code once instead of being held down and executing code.

This is done because if the buttons were held down, the zooming in and out would be too fast and a user may have difficulty adjusting the zoom.

6.1.2 Frame limiting

Limiting frames are very advantageous when it comes to dealing with programs that have graphical rendering. Not all computers may have similar specifications i.e. will work at different speeds to each other; frame limiting ensures that computers with higher specifications are limited to a certain FPS, otherwise, if not limited, the program will be moving so quickly that a user cannot keep up.

```
//handling frames per second
float frameTime = 0,
    prevTime = 0,
    curTime = 0,
    deltaTime = 0;

Uint32 startTime, elapsedTime;
startTime = SDL_GetTicks();

frames = 0, fps = 0;
```

Figure 27 Frame rate variables

6.1.2.1 Setting up frame rate values

Figure 27 shows the variables used to limit the frame rate used in the program. These variables use the SDL Timer to determine the time between frames and are used to calculate the frames per second (FPS) and delay the next frame (which limits the FPS).

6.1.2.2 Code explanation

```
//loop
while (appRun)
{
    //calculating frame time
    frames++;

    //the overall elapsed time
    elapsedTime = SDL_GetTicks() - startTime;

    //setting previous time to current time
    prevTime = curTime;
    //update current time
    curTime = SDL_GetTicks();
```

Figure 28 Start of the loop execution

Every iteration of the program loop is counted as a frame, hence the frames variable is immediately incremented.

The elapsed time is then calculated by calculating how long it has been since the beginning of the program, this is calculated by subtracting the unix time of the current time to the start time.

Previous time is set to become the current time and a new current time is set.

```
//skips first frame to prevent crash
if (elapsedTime)
{
    //calculate time between frames
    deltaTime = (curTime - prevTime) / 1000.0f;

    //update frame time
    frameTime += deltaTime;

    //convert elapsed time to seconds
    double elapsedSec = elapsedTime / 1000;
    fps = frames / elapsedSec;

    //debug fps
    //std::cout << fps << std::endl;

    //controlling frame rate
    if (deltaTime < 1000 / fpsLimit)
    {
        if (fpsLimit != 0)
        {
            //sleep the remaining frame time
            SDL_Delay((1000 / fpsLimit) - deltaTime);
        }
    }
}
```

Figure 29 End of loop execution

Once the timer variables are set, the program runs all the other code which allow the program to work i.e. executes code that allows the user to create a scenario / run a simulation.

Once all the code is executed, the program checks if any frame limiting should be done; this is done by checking if the time to execute the code (deltaTime) reaches the time for one frame (1000 / fpsLimit).

This is done by calculating the deltaTime using the current time and the time from the previous frame; the deltaTime is then compared to see if the frame limit time is not reached.

If the program manages to execute faster than the frame limit value, the program is delayed by how much time is required to meet the aforementioned value.

6.1.3 Main program loop

6.1.3.1 Main.cpp

```
#include "SCMain.h"

int main(int argc, char **argv)
{
    SCMain ZeusSC;
    return 0;
}
```

Figure 30 contents of main.cpp

The entry point for the program begins in the main.cpp file; the file creates an SCMain object in which an SDL Window and inputs are set up.

Figure 30 shows the entirety of main.cpp's contents (for the scenario creator but is still similar in the simulator). It consists of 7 lines of code; SCMain.h is included as it is references in the scope of the main function. The main function only instantiates an SCMain object where the rest of the program is handled.

6.1.3.2 Main.h and SCMain.h

```
SCMain();
~SCMain();

//initialise simulator
void init();

//close app
void close();

//main loop
void mainLoop();

//update main loop
void updateMain();
```

Figure 31 SCMain Functions

Figure 31 shows the functions in SCMain.h; these functions deal with setting up the window and renderer (in the init function) as well as the main program loop, where the input checking is also done.

The main loop is the loop done by the program, and if the loop ends, the program ends. The variable appRun is passed as a reference to the GUI class, where if the variable is set to false, the program ends. The main loop is also where the frame limiting is performed, and each frame loop, the function updateMain is called

The updateMain function updates the renderer so that proper values and graphics are displayed to the screen depending on the user's input. The updateMain function calls the specific GUI class functions that deal with rendering to the screen.

6.2 Zeus Scenario

6.2.1 SCGUI.h

6.2.1.1 Constructor

```
//default constructor and destructor
SCGUI();
~SCGUI();
//main constructor
SCGUI(SDL_Renderer *renderer, int winX, int winY);
```

Figure 32 SCGUI constructor

The SCGUI class has an empty constructor class, as well as a default destructor.

The class also has a constructor where the class is given the renderer as a reference as well as the window width and height.

6.2.1.2 Class functions

```
//main menu bar
void menuBar(SDL_Renderer *renderer, bool &appRun);

//file handling
void open(SDL_Renderer *renderer);
bool loadScenario(std::string filePath, SDL_Renderer *renderer);
void save();
void saveAs();
void saveFile(std::string filepath);

//view country window
void viewCountries();

//new scenario window
void newScenarioWin(SDL_Renderer *renderer);
void resetNewScenario();

//add new country window
void newCountryMenu(int r, int g, int b);
void resetNewCountry();

//edit country window
void editCountryMenu();

//gui rendering
void render(SDL_Window *window, SDL_Renderer *renderer);

//tools
bool doesCountryExist(std::string id);

//ease of access
void eyedropTool();
```

Figure 33 SCGUI functions

Figure 33 shows the functions that deal with the saving and loading of scenario data as well as the functions that serve as a window where users can input data to create / edit countries.

The render function is called from SCMain program loop so that the GUI items are rendered to the window.

The bottom 2 functions are used as background functions that are required to refine how the simulation works i.e. functions that improve user experience.

doesCountryExist is a function used to check if a country exists by checking the unique ID, this is used in multiple places across SCGUI, mainly so correct countries are edited.

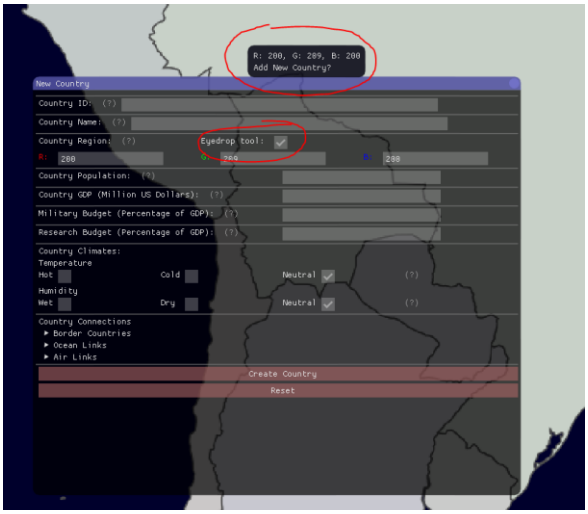


Figure 34 eyedropTool and its effect

The eyedropTool function displays to the screen the RGB value under the mouse pointer; this function is useful for creating new countries and setting their visual region on a map.

Figure 34 shows the eyedropTool option in the New Country window (option also exists in the Edit Country window). The circled object above the option is the tool tip that displays the colour of the country being hovered over by the mouse; the tooltip shows information about whether or not the region already has a country as well as its RGB colour.

6.2.2 Screenshots of the scenario creator

Screenshots are included so that the implementation can be seen

Figure 35 Opening Scenario Creator

The scenario creator opens without anything on screen except for the main menu bar

Figure 37 New Scenario window

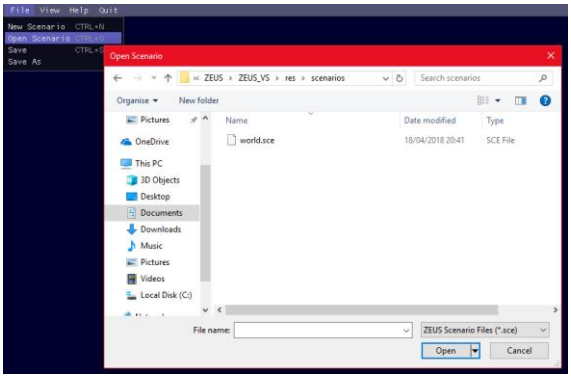


Figure 37 Open Scenario utilises win32 file functions

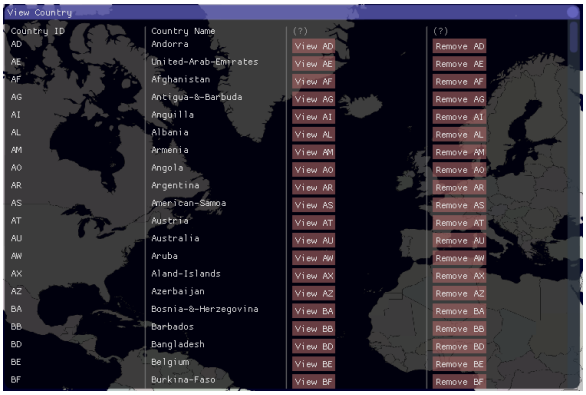


Figure 39 View all countries window

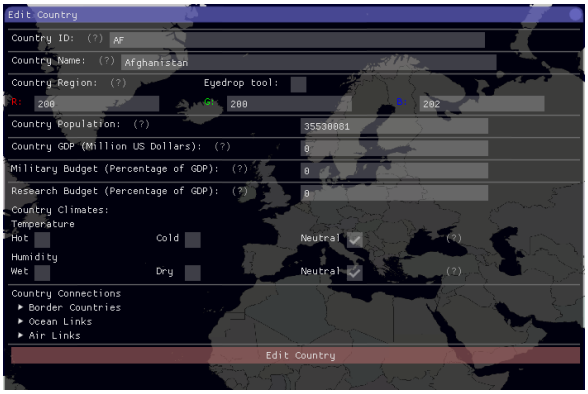


Figure 39 Edit countries window

Note that in Figure 39, the new country window is the same, except that new country window opens with no data in the input boxes.

6.3 **Zeus Main System**

6.3.1 GUI.h

6.3.1.1 Constructor

```
GUI();
GUI(SDL_Renderer *renderer, int winX, int winY);
~GUI();
```

Figure 40 GUI constructors

Similar to Section 6.2.1.1, the GUI class has a default empty constructor and destructor. The GUI class also has a constructor which takes in the renderer, and window width and height as parameters

```
//load country data
bool loadScenario(SDL_Renderer *renderer, std::string filePath);

//draw menu bar
void menuBar(bool &appRun, SDL_Renderer *renderer);

//info box
void infoBox();

//menu bar functions
void newSim(SDL_Renderer *renderer);
void resetNewSim();

void openSim();

void save();
void saveAs();

void editSimVals();
```

Figure 41 GUI functions

6.3.1.2 Class functions

Figure 41 shows the functions used to set up the simulation as well as the sidebar display. The menuBar function is the function that renders the main menu bar on the window.

infoBox is a function that draws the sidebar where information about selected countries are displayed to the screen.

The other functions are used to handle simulation files i.e. save, load or create simulations.

editSimVals is a function that draws he window that users interact with to change the simulation variables.

```
//render items
void render(SDL_Window *window, SDL_Renderer *renderer);
```

Figure 42 GUI render function

Similar to Section 6.2.1.2 the render function is called by the main loop to render to the screen.

6.3.1.3 Simulation function

```
//run simulations
void simulate();
void countrySim(int countryNum);
```

The simulate function is only called when the variable runSim is true. The simulate function calls countrySim, which simulates the spread of the disease within the country object.

Figure 43 Simulation functions

6.3.2 Screenshots of the Simulator

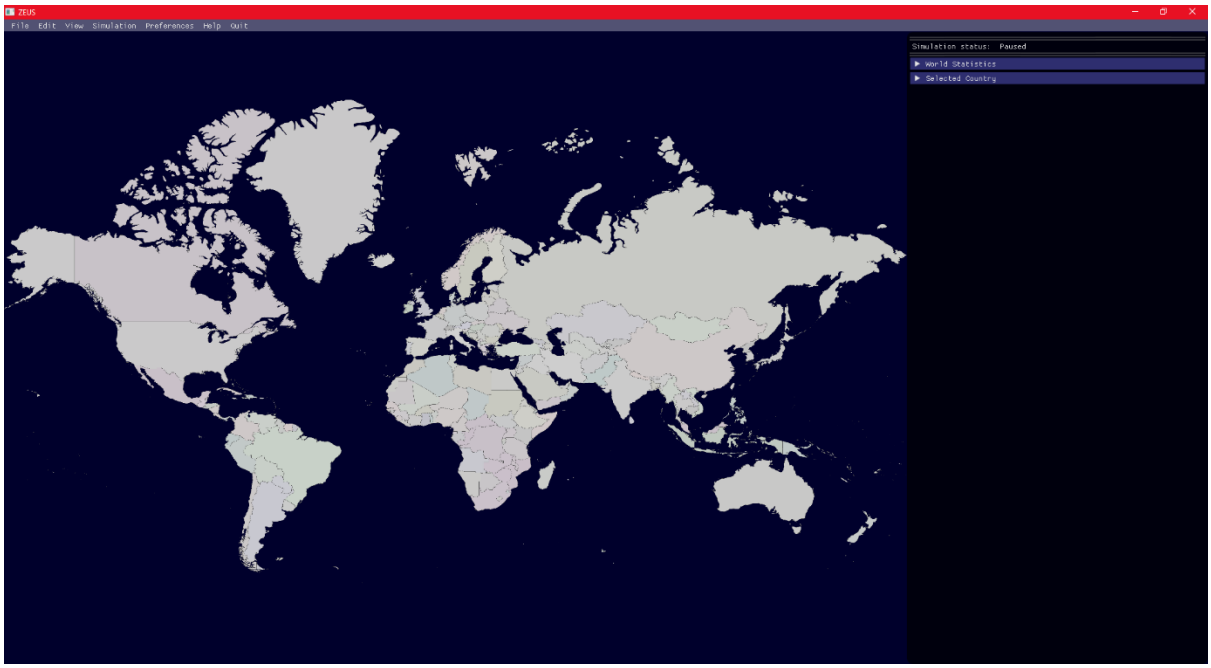
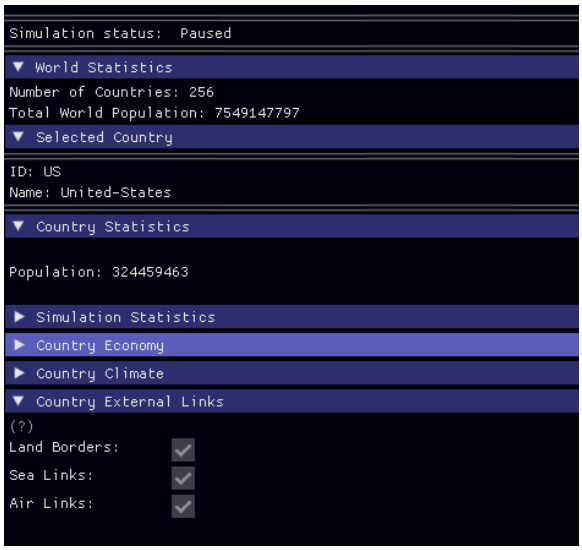


Figure 44 Default Simulator scenario

The simulator will always open with a default scenario, however a user can change the scenario



The info box shows information relevant to the selected country; Figure 45 shows an example of what the info box shows, using the United States data.

Figure 45 InfoBox

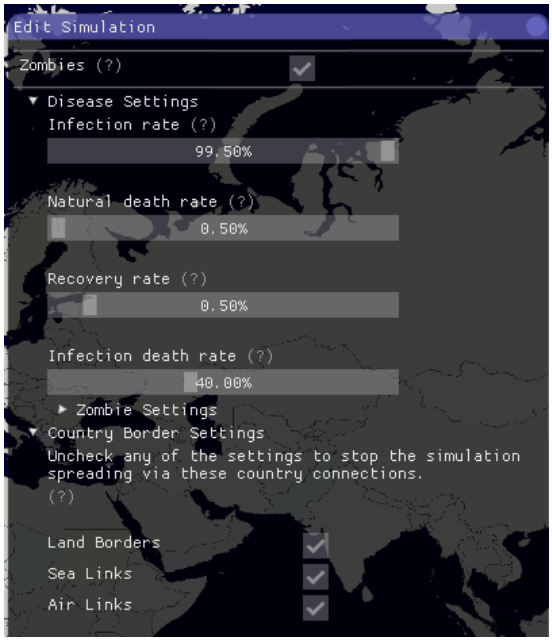


Figure 47 Edit simulation values

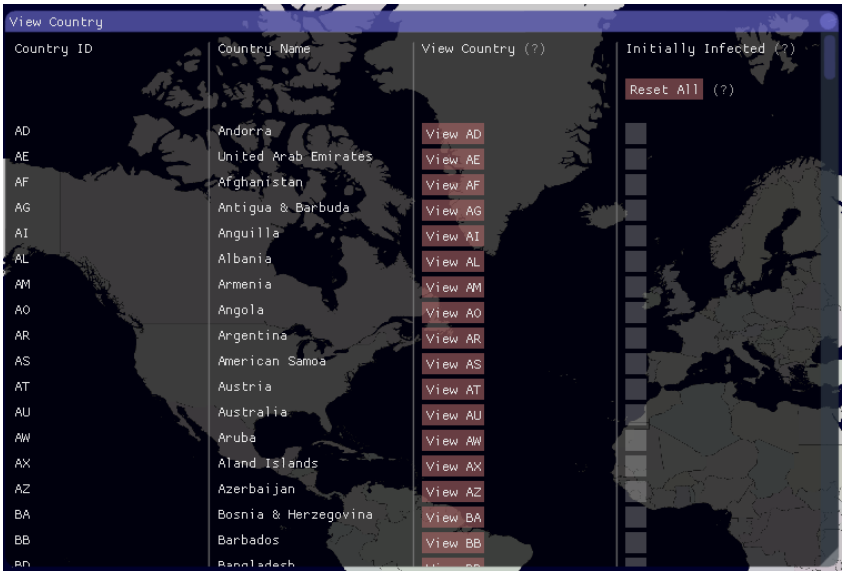


Figure 47 View countries window

Figure 47 is the window where users can set the simulation parameters.

Figure 47 shows the view all countries window where further information about countries can be seen as well as users setting which countries are to be infected.

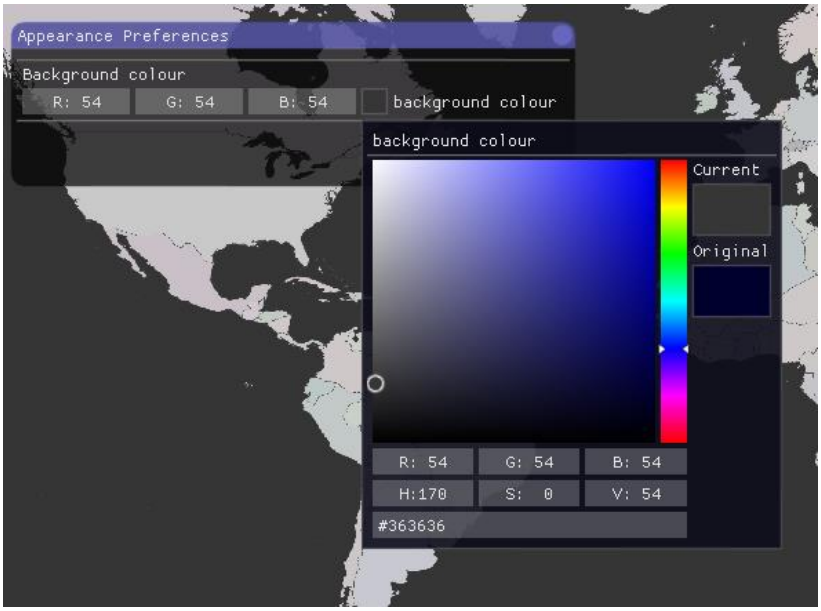


Figure 48 shows the appearance preference which is an option that allows the user to change the colour of the background so that if the simulation map is too hard to differentiate from the background, a user can change it.

Figure 48 Appearance preferences

6.4 **Shared Objects**

Across both the simulator and the scenario creator, there are multiple functions that are usable in both programs. Therefore some source code is in a folder which both projects are able to access so that there is a decrease in code duplication.

6.4.1 Data Handler

```
DataHandler();
DataHandler(std::string filepath);

////////////////////////////////////////////////////////////////////////////////
/**
* SCENARIO
*/
////////////////////////////////////////////////////////////////////////////////
void loadScenario();
void resetAll();

//getters
std::string getSceName();
std::string getTexturePath();
std::vector<Country> getCountryList();
int getCountryCount();
unsigned long long int getTotalPop();
```

Figure 49 DataHandler

The DataHandler class deals with the files that are used across the ZEUS System; this includes the simulation files as well as the scenario files. However, at the time of this submission, the system does not handle simulation files and instead default simulation values are given to the simulator so that is could be demonstrated.

The DataHandler class holds data about the scenario to be loaded i.e. the global values, such as total global population. Each data handler class receives a file path as a parameter, which it then loads the file and extract the country objects (and if implemented, the simulation values).

The DataHandler class also gets the scenario name, map texture path and counts the number of countries loaded, which is information that is displayed in the simulation info on the simulator.

6.4.2 Country Objects

```
//default constructor
Country();
//main constructor
Country(std::string ID, std::string cName, int red, int green, int blue, unsigned long long int pop,
    unsigned long long int gdp, unsigned long long int militarybudget, unsigned long long int researchbudget, int temp, int hum, std::vector<std::string> landBorder,
    std::vector<std::string> sea, std::vector<std::string> air);
~Country();

//getters and setters for the country data
SETTERS AND GETTERS

//get simulation variables
unsigned long long int getHealthyPop() { return healthyPop; }
unsigned long long int getInfectedPop() { return infectedPop; }
unsigned long long int getDeadPop() { return deadPop; }
unsigned long long int getZombiePop() { return zombiePop; }
unsigned long long int getRemovedPop() { return removedPop; }

//reset simulation variables
void resetSimVal();

//initial infection
bool infected = false;

//simulate spread
void simulate(bool simulateZombies, int simFrame,
    float infRate, float naturalDeathRate,
    float recoveryRate, float infDeathRate,
    float zconversionRate, float zdeathRate,
    float zremoveRate, float corpseDecayRate,
    float reanimationRate, bool allowLandInfect, bool allowSeaInfect, bool allowAirInfect,
    std::vector<Country> &countryList);

void receiveInfection(int infectedPassengers);

bool selectedBorder, selectedSea, selectedAir;
```

Figure 50 Country

Figure 50 shows the functions involved in the country object (the setters and getters for the country variables are minimised as the simulation functions are of higher priority).

The constructor for the class takes in parameters from when a country is created via the edit country menu and the new country menu. Getter and setter functions are used to get and set the variables as they are set as private variables.

6.4.2.1 Simulating spread of disease

The spread of diseases are calculated for each country, therefore a function called simulate is called for each country that needs to simulate a spread of the infection. The Coutnry class also has a function called receiveInfection, which is called from another instance of a country object to simulate the fact that infected individuals have arrived from the origin (of disease) country to this country.

```
float getRand(float range1, float range2);


//natural deaths
void naturalDeaths(bool simulateZombies, float naturalDeathRate);

//spread infection
void spreadInfection(bool simulateZombies, float infRate,
    float recoveryRate);

//deaths from infection
void deathRates(bool simulateZombies, float infDeathRate);

//spread to other countries
void infectCountries(bool allowLandInfect, bool allowSeaInfect, bool allowAirInfect, std::vector<Country> &countrylist);

//zombie functions
void zombieSpread(float zconversionRate);
void zombieInfect();
void zombieDecay(float zdeathRate);
void zombieReanimation(float reanimationRate);
```

Figure 51 Country Simulation Functions

Figure 51 shows the functions that are called by the simulate function shown in Figure 50. These functions are called based on randomisation as well as if the country has a population (of healthy, infected or zombie individuals) more than 0.

```
//Control functions
//function to allow zooming
void zoom(int zoomType);

//function to allow panning of the map
void pan(SDL_Point *mPos, int motionX, int motionY);

//ensures the pan is limited within the relevant zone of the texture
void panLimiting();

//mouse over helps identify a country of where there could be one
void mouseOver();
void leftClick();

//shortcut controls
void ctrlS() { save(); };
void ctrlN() { newScenario = true; }
```

Figure 52 SCGUI UX Functions

6.4.3 User Experience functions

There are multiple UX functions used throughout the program to make the use of the ZEUS System more user friendly. The functions provide extra context to users about the actions that they are currently (or prompt users about what they are) doing. The functions also allow users to manipulate the program in a way that is “natural” i.e. act in a way it is expected to.

6.4.3.1 Panning

The pan function allows a user to move the map around while zoomed in; this is so that the user(s) can see the whole scenario map in better detail.

The function is the same in both the scenario creator and the simulator; with the difference being the area of the viewport rectangle.

```
void GUI::pan(SDL_Point *mPos, int motionX, int motionY)
{
    //check if the mouse is within the viewport
    if (SDL_PointInRect(mPos, &vp))
    {
        vpSrc.x += motionX * 2;
        vpSrc.y += motionY * 2;
        //std::cout << "X: " << vpSrc.x << std::endl << "Y: " << vpSrc.y << std::endl;

        //limit the panning so there is no texture streching
        panLimiting();
    }
}
```

Figure 53 Pan function in GUI.h

```
void GUI::panLimiting()
{
    //limit for the left scrolling
    if (vpSrc.x < 0)
    {
        vpSrc.x = 0;
    }
    //limit for up scrolling
    if (vpSrc.y < 0)
    {
        vpSrc.y = 0;
    }
    //limit for scrolling to the right
    if (vpSrc.x > wMapX - vpSrc.w)
    {
        vpSrc.x = wMapX - vpSrc.w;
    }
    //limit for scrolling to the bottom
    if (vpSrc.y > wMapY - vpSrc.h)
    {
        vpSrc.y = wMapY - vpSrc.h;
    }
}
```

Figure 54 Pan limiting function

6.4.3.1.1 Pan limiting

Each call of the pan function also calls the function to limit the pan. This is to prevent the user from losing the focus of the map from the screen i.e. losing sight of the map.

The function uses the viewport x and y coordinates to calculate if the viewport window exceeds the scenario map image size.

```
void GUI::zoom(int zoomType)
{
    //prevents the control if the user is over any ImGui components
    if (!ImGui::IsAnyItemHovered() && !ImGui::IsAnyWindowHovered())
    {
        //store the old zoom
        float oldZoom = zoomVal;

        //determine the zoom type and alter the zoom val
        switch (zoomType)
        {
        case -1:
            //place holder input
            break;

            //zoom in
        case 0:
            if (zoomVal - zoomInterval > minZoom)
            {
                zoomVal -= zoomInterval;
                //std::cout << "zoomed in" << std::endl;
            }
            break;

            //zoom out
        case 1:
            if (zoomVal + zoomInterval <= maxZoom)
            {
                zoomVal += zoomInterval;
                //std::cout << "zoomed out" << std::endl;
            }
            break;

        default:
            std::cout << "Unknown zoom type" << std::endl;
            break;
        }

        //adjust vpSrc
        if (oldZoom - zoomVal != 0)
        {
            vpSrc.w = wMapX * zoomVal;
            vpSrc.h = wMapY * zoomVal;
            vpSrc.x = wMapX / 2 - vpSrc.w / 2;
            vpSrc.y = wMapY / 2 - vpSrc.h / 2;
        }

        //std::cout << zoomVal << std::endl;

        //limit the x and y of the source
        panLimiting();
    }
}
```

Figure 55 Zoom function

6.4.3.2 Zooming

Figure 55 shows the zoom function. The type of zoom is checked by a switch case statement; depending on the zoomType, the zoom value is either decreased if zooming in and vice versa.

The zoom value is used to determine whether the viewport (section of the screen users can see) should increase in size (therefore zooming out) or decrease in size (therefore zooming in).

The panLimiting function is then called to ensure that the viewport remains within the boundary of the texture.

6.4.3.3 Keyboard shortcuts

```
/*
 ctrlN
 function called when the control + N shortcut is used.
 shortcut is to make a new simulation
*/
void GUI::ctrlN() { ... }

/*
 ctrlO
 function called when the control + O shortcut is used.
 shortcut is used to open an existing simulation
*/
void GUI::ctrlO()

    openSimWindow = true;


void GUI::spaceBar()

    run ^= 1;
```

Figure 56 keyboard shortcuts

The keyboard shortcuts are small functions that call other functions relevant to the key press combination detected by the switch case from Section 6.1.1.3.1.4.

Each combination calls a specific shortcut function which in turn calls the relevant function to do what is expected of the shortcut combination.

6.4.3.4 helpMarker

Figure 57 shows the helpMarker in full. The function takes in a C string input and creates an ImGui Text item with the text “(?)” on the part of the screen relevant to where the helpMarker function was called.

```
void helpMarker(const char * desc)
{
    ImGui::TextDisabled("(?)");
    if (ImGui::IsItemHovered())
    {
        ImGui::BeginTooltip();
        ImGui::PushTextWrapPos(450.0f);
        ImGui::TextUnformatted(desc);
        ImGui::PopTextWrapPos();
        ImGui::EndTooltip();
    }
}
```

Figure 57 helpMarker function

When the user puts their mouse over the “(?)” text, a tool tip comes up displaying the C string input. This function is useful for sections where extra information may be required by the user but would not fill the screen with text.

The information in the tool tip prevents the user needing to look for the information required in the user manual, as well as preventing the overload of data to a user.

```
POINT p;
BOOL b;

// Get the current cursor position
b = GetCursorPos(&p);
COLORREF colour;
HDC hDC;

// Get the device context for the screen
hDC = GetDC(NULL);
if (hDC == NULL)
    std::cout << 3;

if (!b)
    std::cout << 2;

// Retrieve the color at that position
colour = GetPixel(hDC, p.x, p.y);
if (colour == CLR_INVALID)
    std::cout << 1;

// Release the device context again
ReleaseDC(GetDesktopWindow(), hDC);
```

Figure 58 Colour detection

6.4.4 *Eyedrop tool*

The eyedrop tool utilises the Win32 API (windows.h include file) functions. The eyedropper function is used to display the RGB colour values of the region where the mouse is hovering over, making it easier for a user to create a scenario.

Source of the code [18].

The first thing the eyedrop tool is to determine the colour that is under the mouse pointer; the mouse position must first be received, after which, the screen that the mouse is hovering over must be received.

Finally, the specific pixel under the mouse is received and its colour is taken.

This RGB value is then printed to the screen as a tooltip; if the country already exists, the country name is displayed as well instead of just the RGB colour.

7. System Testing

Now that a ZEUS prototype has been developed, the prototype needs to be tested to ensure that it meets the set specifications and, in this case, solves the problem outlined in Section 2.

The tests will be performed on a specific software at first; once the test criteria are met by both software, the integration test will be performed. The integration test involves checking whether or not the scenarios created by the scenario creator are being loaded by the simulator.

7.1 **ZEUS Scenario Creator**

Component to test	Input Value	Expected output	Actual output	Passed?	Comment
Application runs	-	The application opens and runs	As expected	Yes	Tested on different computers, and being able to run on all (Used Windows 10 OS)
Create a new scenario	Name: test Map file: default world map	Opens the correct map on screen with no countries	As expected	Yes	
Save scenario	Press the Save button	Should prompt the user to save the file under which name	As expected	Yes	If the file is new, then it should as for a file name to be saved as
	Press save again	Saves to the file straight away without a prompt	As expected		
New country	No input	Error ID and country name is empty	As expected	Yes	
	ID: 1	Error country name is empty	As expected	Yes	
	ID: 1 Country Name: 1	Creates the country with default values	As expected	Yes	
	ID: edit Country Name: edit Land border with 1	Creates a country called edit and both countries now have land border link	As expected	Yes	
Edit country	Rename ID and name of edit to 2	Renames the country id and name	As expected	Yes	
	Add a sea link from 1 to 2	Both are linked via sea	As expected	Yes	

View countries	View country 1	Opens the edit country menu but with the values for country 1	As expected	Yes	
	Remove country 1	Removes country 1 and country 2 loses link to country 1	As expected	Yes	
Quit	Press “Close Application”	Application shuts down	As expected	Yes	

7.2 **ZEUS Main System**

Component to test	Input Value	Expected output	Actual output	Passed?	Comment
Application runs	-	The application opens and runs	Application runs	Yes	Tested on different computers, and being able to run on all (Used Windows 10 OS)
New Simulation	Not implemented				
Open Simulation	Not implemented				
Save / Save As	Not implemented				
Edit Simulation	Unticked zombie	Removed display of zombies from the sidebar and options to edit zombie variables are removed	As expected	Yes	
	Decrease infection rate	Infection rate and natural death rate add up to 100%	As expected	Yes	
	Untick zombie and move recovery rate	Recovery rate and infection death rate add up to 100%	As expected	Yes	
	Zombie sim is ticked, and recovery rate moved around	Zombie conversion rate, infection death rate and recovery rate add up to 100%	As expected	Yes	
	Shift clicking on sliders and adding huge numbers	The slider value is set to its highest value instead of a large number	As expected	Yes	
	Unticking Sea links and running the simulation starting from an island country	Should not spread via the sea	As expected	Yes	

Change scenario	Change the scenario to the previous scenario creator test scenario	Scenario changes	As expected	Yes	
Scenario details	Open the scenario details window	Displays the correct scenario name, country count and total scenario population	As expected	Yes	
Simulation details	Not implemented				
View all countries	Choose China as the start of a disease and then run the simulation	Disease starts from China	As expected	Yes	
Simulation controls	Press Run	Simulation starts	As expected	Yes	
	Press Pause	Simulation stops at where it currently is	As expected	Yes	
	Press Reset	Stops the simulation and resets its values	As expected	Yes	
Appearance preferences	Change the colour of the background to a reddish hue	Colour changes according to what the user sets	As expected	Yes	
Quit	Press “Close Application”	Application closes	As expected	Yes	

8. Conclusions and Evaluation

Referring to the original objective stated in Section 2:

The main problem the project addresses is that there is a lack of simulations regarding the spread of zombies…

… existing simulators also have a wide range of complexities, at a cost of simulation accuracy i.e. the more complex the prompts for input are, the more accurate the simulations; which leads to the secondary problem, which is that the ZEUS System must have a balance where the simulations are semi-realistic without the use of the simulations being too complex.

The main objective of this project was to create software that was capable of simulating a zombie-based disease. In order to do this, existing examples of other simulators were researched to see if they were sufficient in solving the outlined problem, however, after looking at a few examples (including the case study simulators in Section 3.1) the problem has not been solved by already existing examples. The simulators did not solve the problem, however, gave an insight to the possible components that were useful in the development of the ZEUS System. The advantageous components of other simulators were used in the design process of the ZEUS System to see which would benefit the development of the ZEUS System.

Although the prototype ZEUS System is not fully complete, the system is a useful proof of concept. In the future should any work be done on creating a better zombie simulator, the prototype ZEUS System may be of use, as the simulator does simulate the spread of zombies, although not as accurate expected of the simulator. The ZEUS System partially meets the requirements of the first objective, as the simulator is capable of running a zombie simulation

The incomplete ZEUS System managed to meet the requirement of the first problem, which is to develop a zombie-based simulator; however, does not meet the other requirements. The simulator is somewhat hard to use and may need a small read of this document to understand what to do. The zombie simulations that the system produces are also not very realistic, even if using real world data. The second objective was for the ZEUS System to balance simulation accuracy with the ease of use of the system; the system is relatively easy to use, however, this does mean that the simulator is not as accurate as it should be.

9. Future of the project

9.1 **Possible Improvements**

As mentioned in Section 8, the prototype ZEUS System is incomplete. The unfinished components can be finished first i.e. the ability to save and load simulation files are incomplete. Another unfinished planned feature was a colour overlay which visualised which countries have been infected; the only way to see which country(ies) have been infected is to look at the View Countries option in the Simulator.

9.2 **Fixing later discovered bugs**

The project is placed on GitHub. GitHub has a feature in which people with access to the project's repository are allowed to send "Issue" tickets, in which the developer is notified. The issue tickets are where users of the system can raise bugs / problems that have been encountered during the use of the system, which may have been missed during the testing phase.

9.3 **Optimisations**

Looking at the source code for the project, there are multiple functions that have duplicates; this could be removed, therefore making the format of the program code neater.

Another optimisation that the ZEUS System may benefit from is the application of multithreading, especially in loops, so that the execution of simulation calculations are fast and efficient.

10. Glossary

CPU	Central Processing Unit; component in the physical computer, which carries out program instructions.
FPS	Frames Per Second; the number of frames the program renders in a second
Framework	Abstraction of software development where low-level details of a system is pre-programmed and users only need to add code on top to meet software requirements e.g. scripting on the web
GNU	GNU's Not Unix; an operating system consisting of a collection of open source software
GPL	General Public License [19]; a license (mainly used for software) which allows end users the freedom to run, study, share and modify the software
GPU	Graphics Processing Unit; component in the physical computer specialised for rendering of graphics
GUI	Graphical User Interface; a user interface that utilise graphics
IDE	Integrated Development Environment; software application(s) that provide facilities necessary (or advantageous) to developing software
Main	Main System; The main part of the ZEUS System that runs the simulations
RNG	Random Number Generator; allows for randomisation in the application/system
SC	Scenario Creator; the portion of the ZEUS System that handles (and standardises) the scenarios used in ZEUS
UI	User Interface; the part of the software that allows the user and the system to interact
Wiki	A website which allows users to collaboratively edit content using mostly a web browser. Wikis are mostly used to provide context to a specific object / subject. In the case of this paper, a wiki is used to explain the project and its components

11. Appendices and References

11.1 **Project Initiation Document**

(on next page)

Individual Project (CS3IP16)

Department of Computer Science
University of Reading

Project Initiation Document

PID Sign-Off

Student No.	**24008064**
Student Name	**Czar Ian Echavez**
Email	**c.i.echavez@student.reading.ac.uk**
Degree programme (BSc CS/BSc IT)	**BSc Computer Science**
Supervisor Name	**Prof. Atta Badii**
Supervisor Signature	
Date	

SECTION 1 – General Information

Project Identification

1.1	**Project ID** (as in handbook)
	370
1.2	**Project Title**
	Zombie Simulation
1.3	**Briefly describe the main purpose of the project in no more than 25 words**
	Creation of a simulation software that helps model how a realistic (or custom) zombie infestation would spread across the world

Student Identification

1.4	**Student Name(s), Course, Email address(s)** e.g. Anne Other, BSc CS, a.other@student.reading.ac.uk
	Czar Ian Echavez, BSc Computer Science, c.i.echavez@student.reading.ac.uk

Supervisor Identification

1.5	**Primary Supervisor Name, Email address** e.g. Prof Anne Other, a.other@reading.ac.uk
	Prof. Atta Badii
1.6	**Secondary Supervisor Name, Email address** Only fill in this section if a secondary supervisor has been assigned to your project

Company Partner (only complete if there is a company involved)

1.7	**Company Name**
1.8	**Company Address**
1.9	**Name, email and phone number of Company Supervisor or Primary Contact**

SECTION 2 – Project Description

2.1 **Summarise the background research for the project in about 400 words. You must include references in this section but don't count them in the word count.**

Research will be split into 2 categories; the first involves research that is outside of the realms of Computer Science, such as Human Geography, Epidemiology and Pathology. The second involves the development of software, which includes UIs, pre-existing software linked to the project and the optimisations that could be applied to the software

Research will include how normal (non-zombie) infections spread; this will generally be considering the mathematics involved with epidemic theory [1][2] as well as the different pathogens which can act as the cause of zombification, especially the terrifying cordyceps fungi which already cause insects to behave in a zombielike manner [3].

Another background research needed is the population geography of humans around the world. This is to properly model how the infection would spread, since people are used as infection vectors (and hosts) to propagate the zombie infection. This part would need to look through population censuses around the world.

As mentioned before, there are already existing examples of the software such as [4] which has a user-friendly UI, however the algorithm used for the spread of zombies is very simplified and uses the Gillespie algorithm which is more commonly used with chemical reactions (usually where the initial amount must end with the same amount i.e. zombies to people amount). Another simulation that is linked to the project is the GLEAM Simulator [5]; the simulator allows 2D and 3D visualisation as well as a builder for simulation scenarios.

(Not for zombie simulation but the idea in this project [6] can be transferred into this project)

The final but most important research that must be done is how to develop the simulation, that includes the language, the libraries and IDE to use. I have experience using SDL as a 2D rendering library which will help with the development of the simulation visualisation. However, I would also like to see if it is possible to visualise the simulation in 3D.

[1] http://post.queensu.ca/~ja9/My_Homepage_Files/Download/Epidemic%20Theory%20and%20Group%20Violence.pdf
[2] https://dspace.library.uu.nl/bitstream/handle/1874/8591/heesterbeek_96_concept_epidemic.pdf?sequence=3
[3] http://journals.plos.org/plosone/article/file?id=10.1371/journal.pone.0036352&type=printable
[4] https://mattbierbaum.github.io/zombies-usa/
[5] http://www.gleamviz.org/simulator/
[6] https://github.com/Hopson97/Empire

2.2 **Summarise the project objectives and outputs in about 400 words.**
These objectives and outputs should appear as tasks, milestones and deliverables in your project plan. In general, an objective is something you can do and an output is something you produce – one leads to the other.

Tasks to complete:

Background Research:

- Research how real-life diseases spread around the world, particularly pandemic level diseases
- Research into epidemic theory; including r0, incubation periods of diseases, and various kinds of transmission vectors
- Research of pathogens
- Find data about population numbers across multiple countries
- Look at pre-existing software and find some ideas which can be implemented to the project

The research will be used in analysis to help with the designing of how the product will look like as well as how the prototype will function.

Analysis and design:

- Parse through research to find information relevant to the project.
- Turn the equations into pseudocode functions i.e. make some pseudocode
- Create flow charts to show the processes which would be done by the program
- Create UML diagrams to plan the functions and classes which will be used in the project

The analysis and design makes the development of the software much easier since the time spent creating the program is not wasted on going back and forth between the files trying to see what has been already done. Parsing through the research narrows down the resources and makes the finding of relevant information easier. Pre-converting the equations to pseudocode makes it easier to create the final program code. Creating both the flowcharts and the UML is the design outline which will be used during development as the layout of how the classes and their functions should be.

Develop prototype:

- Create the prototype based on the designs
- optimisations

Testing evaluation/validation:

- Test the prototype:
    - User inputs
    - File inputs
    - Saving/loading data
    - Graphics rendering
    - Performance testing, possibly on different machines

Assessments:

- Report documentation
- Poster

2.3	**Initial project specification - list key features and functions of your finished project.** Remember that a specification should not usually propose the solution. For example, your project may require open source datasets so add that to the specification but don't state how that data-link will be achieved – that comes later.
	UI: - Allows user to manipulate the simulation - Visualises the simulation in either 2D or 3D - Allows the user to choose whether to display the simulation while running Accuracy: - The simulation is reasonably accurate. (can be tested by comparing simulations to real life events) - The software follows the algorithms and the relevant calculations as well as gives correct outputs to the user inputs. Inputs: - Simulation can save scenarios as well as load previous scenarios - Simulation can be changed while running i.e. the user can change infection statistics while the simulation is running so that the simulation can be manipulated live
2.4	**Describe the social, legal and ethical issues that apply to your project. Does your project require ethical approval?**
2.5	**Identify and lists the items you expect to need to purchase for your project. Specify the cost (include VAT and shipping if known) of each item as well as the supplier.** e.g. item 1 name, supplier, cost
	- Possible printing and binding costs (~£5-£10)
2.6	**State whether you need access to specific resources within the department or the University e.g. special devices and workshop**

SECTION 3 – Project Plan

3.1	**Project Plan** Split your project work into sections/categories/phases and add tasks for each of these sections. It is likely that the high-level objectives you identified in section 2.2 become sections here. The outputs from section 2.2 should appear in the Outputs column here. Remember to include tasks for your project presentation, project demos, producing your poster, and writing up your report.

Task No.	Task description	Effort (weeks)	Outputs
1			
1.1	**Background Research**	3	…
1.2	…		
	All research done at the same time (this is to ensure that the information found can be easily cross checked across different sources)	3	All the information needed for how the program should calculate the spread of the zombies to make a realistic simulation
2	**Analysis and design**	4	
2.1	…		...
2.2	…		…
	Parse research data	0.5	Relevant research data to be used for analysis and design
	Creation of pseudocode	0.5	Pseudocode usable for the development later
	Creation of flowcharts for the various parts of the program	1.5	Flowcharts that show all the processes involved in the running of the program
	Creation of the UML	1.5	UML which will show the layout of the entire program, including the classes and function which will be involved
3	**Develop prototype**	10	
3.1	…	…	…
3.2	…	…	…
	Creation of the entire prototype	8	A fully functioning prototype which can be used to demonstrate the project
	Project demonstration	n/a	
	Optimisations on the programs and code clean-up	2	A cleaned-up version of the prototype which should make the prototype run smoother
4	**Testing, evaluation/validation**	1.5	
4.1	unit testing		…
4.2	…	…	…
	Test user and file (loading and saving) inputs	0.5	Less buggy user and file inputs
	Graphics tests	0.5	Less buggy graphics
	Performance tests	0.5	Allows for the developer to see the minimum specifications to run the program
5	**Assessments**	3.5	
5.1	produce poster	0.5	Poster
5.2	write-up project report	2	Project Report
	Final demonstration and presentation	1	
TOTAL	**Sum of total effort in weeks**	**22**	

SECTION 4 - Time Plan for the proposed Project work

For each task identified in 3.1, please *shade* the weeks when you'll be working on that task. You should also mark target milestones, outputs and key decision points. To shade a cell in MS Word, move the mouse to the top left of cell until the curser becomes an arrow pointing up, left click to select the cell and then right click and select 'borders and shading'. Under the shading tab pick an appropriate grey colour and click ok.

11.1.1.1.2 START DATE: ../../…. <enter the project start date here>

11.1.1.1.3 Project Weeks

11.1.1.1.1 Project stage	0-2	2-4	4-6	6-8	8-10	10-12	12-14	14-16	16-18	18-20	20-22	22-24	24-26
1 Background Research													
All research to be done in 3 weeks													
2 Analysis/Design													
Parse research data													
Creation of pseudocode													
Creation of flowcharts													
Creation of UMLs													
3 Develop prototype.													
Creating the entire prototype													
Project demonstration													
Program optimisations and code clean-up													
4 Testing, evaluation/validation													
Test user and file (loading and saving) inputs													
Graphics tests													
Performance tests													
5 Assessments													
Project poster													
Report write up													

Project demonstration and presentation													

RISK ASSESSMENT FORM

Assessment Reference No.		**Area or activity assessed:**	
Assessment date			
Persons who may be affected by the activity (i.e. are at risk)			

SECTION 1: Identify Hazards - *Consider the activity or work area and identify if any of the hazards listed below are significant (tick the boxes that apply).*

1.	Fall of person (from work at height)		**6.**	Lighting levels		**11.**	Use of portable tools / equipment		**16.**	Vehicles / driving at work		**21.**	Hazardous fumes, chemicals, dust		**26.**	Occupational stress	✓
2.	Fall of objects		**7.**	Heating & ventilation		**12.**	Fixed machinery or lifting equipment		**17.**	Outdoor work / extreme weather		**22.**	Hazardous biological agent		**27.**	Violence to staff / verbal assault	
3.	Slips, Trips & Housekeeping		**8.**	Layout , storage, space, obstructions		**13.**	Pressure vessels		**18.**	Fieldtrips / field work		**23.**	Confined space / asphyxiation risk		**28.**	Work with animals	
4.	Manual handling operations		**9.**	Welfare facilities		**14.**	Noise or Vibration		**19.**	Radiation sources		**24.**	Condition of Buildings & glazing		**29.**	Lone working / work out of hours	
5.	Display screen equipment	✓	**10.**	Electrical Equipment	✓	**15.**	Fire hazards & flammable material		**20.**	Work with lasers		**25.**	Food preparation		**30.**	Other(s) - specify	

SECTION 2: Risk Controls - *For each hazard identified in Section 1, complete Section 2.*

Hazard No.	Hazard Description	Existing controls to reduce risk	**Risk Level** (tick one)			Further action needed to reduce risks *(provide timescales and initials of person responsible)*
			High	Med	Low	
5	Long uses of display equipment can cause eyesores and nausea	5 – 15 minutes breaks between 1 hour of use of the computer			✓	
10	Excessive electrical usage can cause fire or equipment damage	Using surge protected sockets as well as ensuring all plugs have the correct fuse type for each plug			✓	
26	Too much stress can cause psychological trauma	n/a		✓		
Name of Assessor(s)			**SIGNED**			
Review date						

Health and Safety Risk Assessments – continuation sheet

Assessment Reference No	
Continuation sheet number:	

SECTION 2 continued: Risk Controls

Hazard No.	Hazard Description	Existing controls to reduce risk	**Risk Level** (tick one)			Further action needed to reduce risks *(provide timescales and initials of person responsible for action)*
			High	Med	Low	
Name of Assessor(s)			**SIGNED**			
Review date						

11.2 **Project Logbook**

Final Year Project Log Book

Czar Ian Echavez

Zombie Epidemic Universe Simulation

Computer Science BSc

Academic Year 2017-2018

Date	Week Num	Task (Phase Name)	Task Information (Sub phase name)	Comments	Result of activity	Status of activity	Future Notes
29/09/2017	0	Other	Project Initiation Document	Project Initiation document has been submitted for review		Finalised	
04/10/2017	1	Background research	Looking at existing research papers on zombie mathematics	found a good starting point for the formulas the simulation can be based on		In Progress	source: http://mysite.science.uottawa.ca/rsmith43/Zombies.pdf
05/10/2017	1	Background research	""			Completed	
06/10/2017	1	Background research	Different parameters which could be used in the simulation			In Progress	
07/10/2017	1	Background research	""		refining the formula and brainstorming possible parameters for the simulation	Completed	Separate classes of removed zombies from destroyed zombies, because some corpses cannot be reanimated if destroyed. As well as figure out how to make the zombies decay.
09/10/2017	2	Background research	Different type of zombies in media	2 types of zombies, slow and fast; speed and durability of		In Progress	

				zombies as parameters			
10/10/2017	2	Background research	""		2 more parameters that can be implemented	Completed	
11/10/2017	2	Background research	Other factors in the simulation - the transmission parameter			In Progress	
12/10/2017	2	Background research	""			Completed	
12/10/2017	2	Background research	Other factors in the simulation - natural death rate parameter			In Progress	
13/10/2017	2	Background research	""	Useful website, data.worldbank.com which has data about countries		Completed	CSV file format for the data
13/10/2017	2	Other	Meeting with Atta Badii, project supervisor	Talking about ideas for the design stage	Starting to think of UML designs, colour schemes, especially for colour blind users	Completed	

14/10/2017	2	Background research	Zombie conversion ratio			In Progress	
16/10/2017	3	Background research	""			Completed	
18/10/2017	3	Background research	Researching other development environments and programming languages	Choosing to use C++ in a Visual Studio 2017 IDE	Chosen a programming language and an IDE to create the project in	Completed	Research SDL and SFML
19/10/2017	3	Background research	Research of SFML, SDL and DirectX	I already have experience with SDL, SFML seems easier to integrate with other libraries. DirectX looks like hell		Completed	Possibly going to end up with SDL, since no other graphics libraries may be used
21/10/2017	3	Background research	Researching how to statically link the SDL libraries	Static linking ensures that the release build i.e. end application does not have too many libraries, because they'd be embedded in the executable	Improves quality of life for end user	Completed	apply to the application straight away and don’t change until te end of the project

22/10/2017	3	Project development	Initiating the project by creating the Visual Studio project and linking all SDL files	Set up the project so that the release file only needs one dll	set up the project	Completed	
23/10/2017	4	Research analysis (parsing research)	Turning the notes into a document in the project documentation	A lot of notes to parse through		In Progress	continue parsing the extra parameters that can be included in the project
25/10/2017	4	Research analysis (parsing research)	""			Completed	Create use cases
26/10/2017	4	Project development	Get the simulation to make a window			Completed	Make the simulator to have resizable windows
27/10/2017	4	Project documentation	Setting up the project documentation and completing the project abstract			Completed	
29/10/2017	4	Project development	Restarting with JavaFX and Java language	Too many libraries to handle with SDL and C++, since GUI development requires multiple libraries with many dependencies		In Progress	

02/11/2017	5	Project development	Returning to C++ And VS			Completed	
03/11/2017	5	Project development	Cleaning up code to set up ImGui			Completed	
07/11/2017	6	Project development	Testing ImGui components to see what can be added to the project			In Progress	
14/11/2017	7	Project development	Fixing minor problems which cause UX annoyances			Completed	
21/11/2017	8	Project development	Adding project files to github repository to help keep track of the changes in the project			Completed	
24/11/2017	8	Project development	Setting up textures so that a map is displayed			Completed	
27/11/2017	9	Project development	Implementation of a zoom function	A break in ImGui exists where the menu bar does not work if the user begins with their mouse on the app window		In Progress	
29/11/2017	9	Project development	Implementation of frame limiting			In Progress	
01/12/2017	10	Other	Took a break for a few days	I need a rest			

04/12/2017	11	Project development	Attempt at fixing and therefore improving the zoom function			In Progress	
07/12/2017	11	Project development	Fixed the menu bar problem by removing a line: SDL_GetCurrentDisplayMode			Completed	
08/12/2017	11	Project development	Setting up the program to be able to parse data	country data which will be used to load the country data		Completed	
09/12/2017	11	Project development	Successfully able to load external data	removed zoom and pan because it broke, oops		In Progress	
11/12/2017	12	Project development	Created an image which is now the default map. Countries are now identifiable via mouse click			Completed	
12/12/2017	12	Project development	Rethinking how there should be a separate program to create scenarios; Sub project started for the scenario creator			In Progress	
13/12/2017	12	Project development	Panning and zooming reimplemented			In Progress	

14/12/2017	12	Project development	Continued work on scenario creator			In Progress	
20/12/2017	13	Other	Taking the week off, holiday and rest				
04/01/2017	15	Project development	Resuming with work	Worked on GUI inputs and outputs, user experience and interface work			
08/01/2017	16	Project development	Standardising the scenario file types			In Progress	
11/01/2017	16	Project development	Changing some data types to unsigned long long to be able to handle huge numbers because of population	Added some opengl libraries to the dependencies file so that I can work on the project on other computers, especially the labs which don't seem to have the windows sdk installed		Completed	
11/01/2017	16	Project documentation	Feedback form for supervisor			Completed	

13/01/2017	16	Project development	Implementation of data saving	Can now save scenarios	some user friendlyness fixes, i.e. auto linking of countries, input validations etc	In Progress	
14/01/2017	16	Project development	Mini test of previous additions			In Progress	
15/01/2017	17	Project development	Working on loading scenario data			In Progress	
16/01/2018	17	Project development	Progress on loadScenario function			In Progress	
18/01/2018	17	Project development	loadScenario function finished			Completed	Make this function be usable in the simulator as well
13/02/2018	21	Project development	Finished Scenario Creator			Completed	
14/03/2018	25	Project documentation	starting to create project poster			In Progress	
24/03/2018	26	Project documentation	Poster completed and submitted			Completed	
26/03/2018	26	Project documentation	Creation of the project report	Most headings set out		In Progress	

26/03/2018	27	Other	Added MIT License to the project on GitHub			Completed	
27/03/2018	27	Input and output testing	Created a release build of the incomplete system for other people to check currently working systems			Completed	
29/03/2018	27	Project documentation				In Progress	
30/03/2018	27	Project development	minor adjustments to Scenario Creator and Main simulator			Completed	
31/03/2018	27	Project documentation	Progress on report	Halfway done with the report		In Progress	
05/04/2018	28	Project documentation	Progress on report			In Progress	
05/04/2018	28	Project documentation	Progress on simulator	Most inputs are set up		In Progress	
06/04/2018	28	Project development	Explaining extra tools used in documentation			Completed	
09/04/2018	29	Project documentation	Progress on report			In Progress	
09/04/2018	29	Project documentation	Progress on simulator			In Progress	

10/04/2018	29	Project development	Simulator allows users to switch between scenarios in the simulation			Completed	
10/04/2018	29	Project development	Users can choose which country the infection spreads from			Completed	
11/04/2018	29	Project development	More simulation settings			In Progress	
13/04/2018	29	Project development	Minor tweaks in Scenario creator to reflect changes in main simulator			Completed	
13/04/2018	29	Project development	Sliders not used to input instead of typing values in			Completed	
13/04/2018	29	Project documentation	Progress on report			In Progress	
17/04/2018	30	Project development	Simulator can now run epidemic simulations, but not zombies yet			In Progress	
17/04/2018	30	Other	Setting up presentation for the presentation and demonstration of the project	Noticed that this was late and also accidentally sent the wrong file to blackboard		Completed	

18/04/2018	30	Final demonstration/presentation	Demonstrate the project to a panel	It went well, presentation shown to panel is different to the blackboard version		Completed	
19/04/2018	30	Project documentation	Finishing the report			In Progress	
20/04/2018	30	Other	This is the last entry on the log book, if you want a complete version of every single commit, check out this project on blackboard	Note: Anything commited after 20/02/2018 is no longer part of the submitted document and is me working on improving the project outside of this		Completed	

11.3 **References**

References will be in the format:

<Object being referred to> [<Reference type: Book, Journal, Online, etc>] [<Access date and time *]
<If online, the link to the reference || otherwise the book ISBN or journal ID>

* time in GMT

---

[1] Omar Cornut's GitHub page [Online] [Available as of April 2018]
https://github.com/ocornut

[2] ZEUS System GitHub repository [Online] [Available as of April 2018]
https://github.com/Czar-Ec/ZEUS

[3] GLEAM Simulator [Online] [Accessed 26 March 2018 09:33]
http://www.gleamviz.org/simulator/

[4] GLEAM Simulator version 6.8 Manual [Online] [Accessed 26 March 2018 20:10]
http://www.gleamviz.org/simulator/GLEAMviz_client_manual_v6.8.pdf

[5] Zombietown USA Source (Github page) [Online] [Accessed 26 March 2018 20:43]
https://github.com/mattbierbaum/zombies-usa

[6] Zombietown USA Browser Demo [Online] [Accessed 26 March 2018 20:45]
http://mattbierbaum.github.io/zombies-usa/

[7] WHEN ZOMBIES ATTACK!: MATHEMATICAL MODELLING OF AN OUTBREAK OF ZOMBIE INFECTION [Online] [Accessed 27 March 2018 02:26]
https://mysite.science.uottawa.ca/rsmith43/Zombies.pdf

[8] CIA World Factbook data on world Health Expenditures [Online] [Accessed 29 March 2018 17:20]
https://www.cia.gov/library/publications/the-world-factbook/rankorder/2225rank.html

[9] Visual Studio IDE download page [Online] [Accessed 04 April 2018 16:45]
https://www.visualstudio.com/vs/

[10] Visual Studio Professional and Enterprise edition pricing [Online] [Accessed 04 April 2018 17:00]
https://www.visualstudio.com/vs/pricing/

[11] Git main web site [Online] [Accessed 04 April 2018 18:00]
https://git-scm.com/

[12] Notepad++ Webpage [Online] [Accessed 04 April 2018 23:30]
https://notepad-plus-plus.org/

[13] A Proposal to Add 2D Graphics Rendering and Display to C++ [Online] [Accessed 06 April 2018 04:39]
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n4021.pdf

[14] SDL library homepage [Online] [Accessed 06 April 2018 16:46]
https://www.libsdl.org/

[15] Sam Lantinga LinkedIn page [Online] [Accessed 06 April 2018 04:54]
https://www.linkedin.com/in/sam-lantinga-02771b10/

[16] GNU Lesser General Public License [Online] [Accessed 06 April 2018 05:15]
https://www.gnu.org/licenses/lgpl-3.0.en.html

[17] ImGUI (dear imgui) GitHub page [Online] [Accessed 08 April 2018 09:15]
https://github.com/ocornut/imgui

[18] Stack overflow question about colour detection [Online] [Accessed 13 April 2018 03:19]
https://stackoverflow.com/questions/3078919/how-do-i-get-the-pixel-color-under-the-cursor

[19] GPL (General Public License) Web page [Online] [Accessed 04 April 2018 18:20]
https://www.gnu.org/licenses/gpl-3.0.en.html