

ZEUS  
Zombie Epidemic Universe Simulator

Czar Ian Echavez

(24008064)

BSc Computer Science

Project ID: 370

Module: SE3IP11

Supervisor: Professor Atta Badii

# Abstract

ZEUS is a simulator that can be used to create a visual model of 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.

# 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 I would like to thank is Omar Cornut [[[1]](#endnote-1)] for the development of the dear imgui (ImGUI) header library used extensively during the development of the ZEUS System.

A group I would like to also thank are the multiple collaborators that have contributed to the development (and maintenance) of the Simple Direct-Media Layer (SDL) [14] graphics library.

(Use of ImGUI under the MIT License)

(Use of SDL under the zlib and LGPL license)

I would also like to thank the following people for testing the ZEUS system:

* Tiago Silva
* Doreen Crump

Contents

[Abstract 2](#_Toc511525716)

[Acknowledgements 3](#_Toc511525717)

[1. Introduction 6](#_Toc511525718)

[2. Requirements and Analysis (Literature Review) 7](#_Toc511525719)

[2.1 Case Studies (Existing examples) 7](#_Toc511525720)

[2.1.1 GLEAM Simulator 7](#_Toc511525721)

[2.1.2 Zombietown USA 9](#_Toc511525722)

[2.2 Simulation Mechanisms 10](#_Toc511525723)

[2.2.1 Mathematical model of a zombie disease 10](#_Toc511525724)

[3. Design Specification 12](#_Toc511525725)

[3.1 ZEUS System model 12](#_Toc511525726)

[3.1.1 Breakdown of each variable 13](#_Toc511525727)

[3.2 Use Cases 15](#_Toc511525728)

[3.3 User Interface 18](#_Toc511525729)

[4. Development Tools 20](#_Toc511525730)

[4.1 Visual Studio IDE 20](#_Toc511525731)

[4.2 Git, GitHub and GitHub Desktop 20](#_Toc511525732)

[4.3 Notepad++ 21](#_Toc511525733)

[4.4 External dependencies / libraries 21](#_Toc511525734)

[4.4.1 SDL (Simple Direct-Media Layer) 22](#_Toc511525735)

[4.4.2 ImGUI (Immediate mode GUI) 23](#_Toc511525736)

[5. System Development 24](#_Toc511525737)

[5.1 Window and Renderer 24](#_Toc511525738)

[5.1.1 SDL variables 24](#_Toc511525739)

[5.1.2 Frame limiting 28](#_Toc511525740)

[5.2 Zeus Scenario 29](#_Toc511525741)

[5.2.1 Main.cpp 29](#_Toc511525742)

[5.2.2 SCMain.h 29](#_Toc511525743)

[5.3 Zeus Main System 29](#_Toc511525744)

[5.4 Shared Objects 29](#_Toc511525745)

[5.4.1 Data Handler 29](#_Toc511525746)

[5.4.2 Country Objects 29](#_Toc511525747)

[5.4.3 User Experience functions 30](#_Toc511525748)

[5.4.4 Default Scenario 31](#_Toc511525749)

[5.4.5 Default Simulation 31](#_Toc511525750)

[6. System Testing 32](#_Toc511525751)

[6.1 ZEUS Scenario Creator 32](#_Toc511525752)

[6.2 ZEUS Main System 32](#_Toc511525753)

[6.3 Integration Tests 32](#_Toc511525754)

[7. Usability evaluation 33](#_Toc511525755)

[8. Project outcomes (lessons learned) 34](#_Toc511525756)

[9. Conclusions and Evaluation 35](#_Toc511525757)

[9.1 Project Conclusion 35](#_Toc511525758)

[9.2 Project Reflection 35](#_Toc511525759)

[10. Future of the project 36](#_Toc511525760)

[10.1 Possible Improvements 36](#_Toc511525761)

[10.2 Fixing later discovered bugs 36](#_Toc511525762)

[11. Glossary 37](#_Toc511525763)

[12. Appendices and References 39](#_Toc511525764)

# Introduction

This paper details the research, design and development of the Zombie/Epidemic Universe Simulator (ZEUS) System [[[2]](#endnote-2)].

# Requirements and Analysis (Literature Review)

## Case Studies (Existing examples)

### GLEAM Simulator

The GLEAM (Global Epidemic and Mobility Model) Simulator [[[3]](#endnote-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.

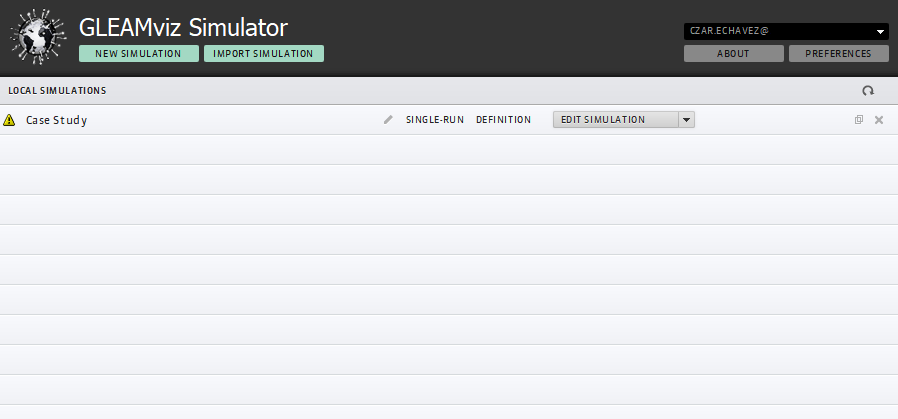


Figure Simulation Manager

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 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.

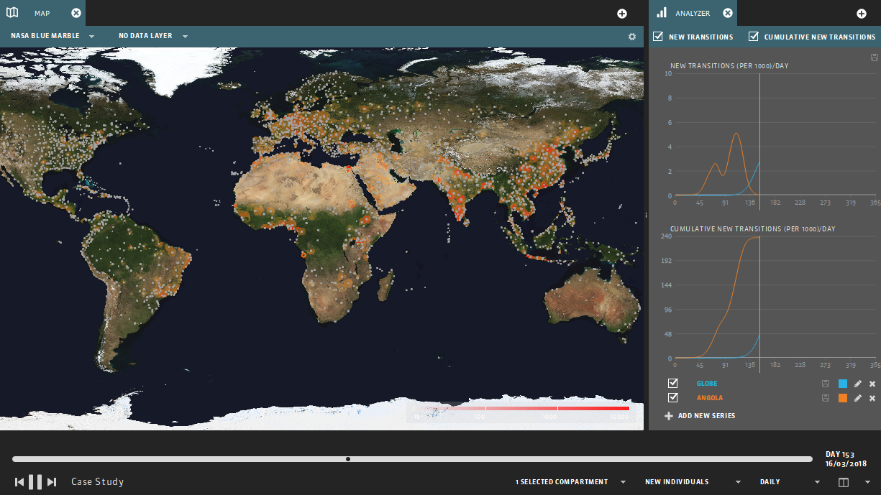


Figure Simulation Visualiser

#### 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]](#endnote-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.

### Zombietown USA

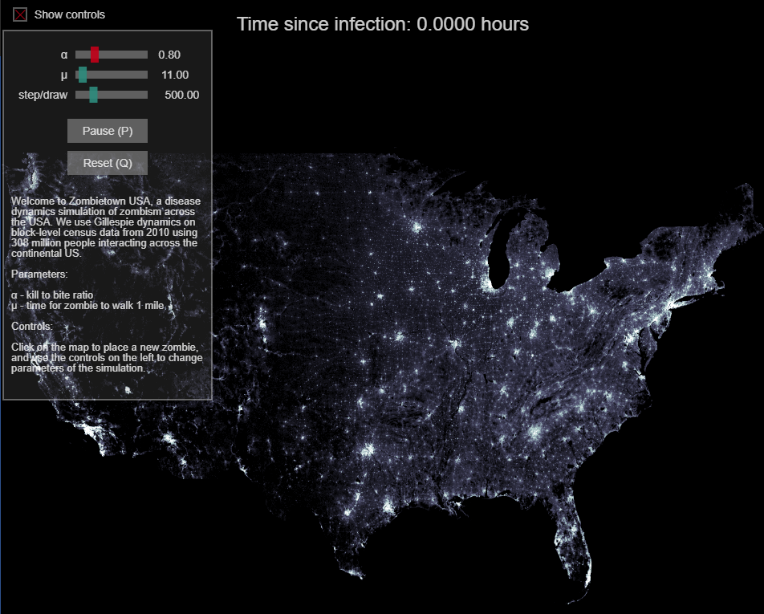


Figure Zombietown USA demo

Zombietown USA [[[5]](#endnote-5)][[[6]](#endnote-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.

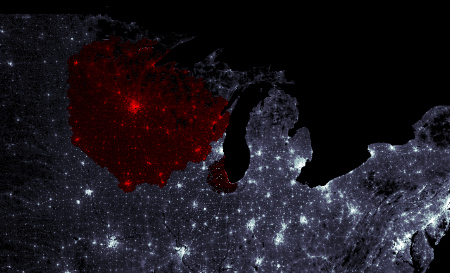


Figure Infection Spread

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 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.

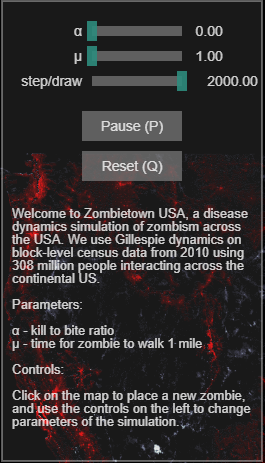


Figure Simulator Variables

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.

#### 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.

## Simulation Mechanisms

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

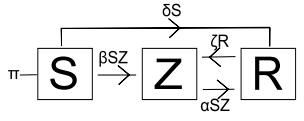


Figure Basic infection model

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.

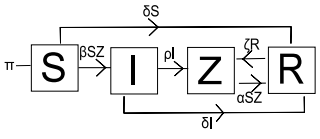


Figure SIZR Model (with latent infection)

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.

# Design Specification

## ZEUS System model

Figure Proposed ZEUS Model



Figure 9 is the proposed simulation model for ZEUS; this model is based from the SIZR model discussed in 2.2.1 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.

### 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 |
| ­α1 | 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]](#endnote-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) |
| δ1 | Decay rate | 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.  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. |
| δ2 | 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). |
| Δ1 | Zombie elimination rate (non-permanent) | Similar to δ2 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 δ2. |
| 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. |
| ζ1 | 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 |

## 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 | |

## User Interface

#### ZEUS Scenario Creator

##### 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.

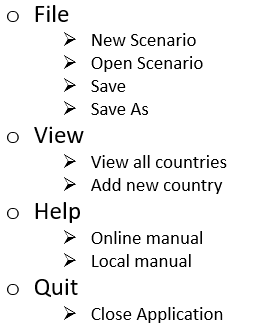


Figure Scenario Creator

Main menu bar hierarchy

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.

“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.

#### ZEUS Main System

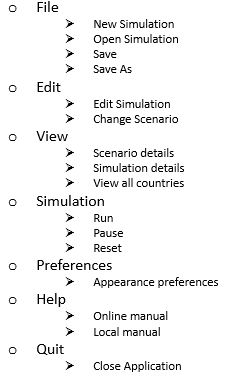


Figure ZEUS Main System

Main menu bar hierarchy

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

# 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.

## Visual Studio IDE

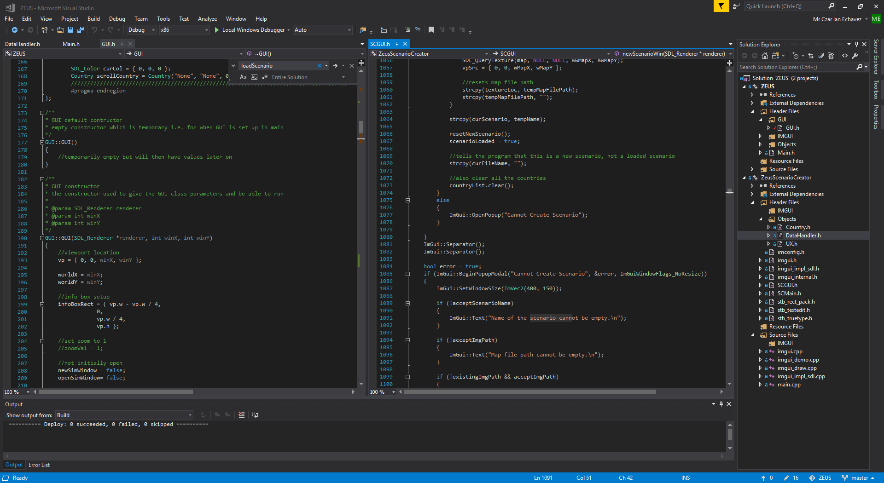


Figure Visual Studio Layout while developing ZEUS

Visual Studio (Community) 2017 [[[9]](#endnote-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]](#endnote-10)]. The software being developed is planned to remain open source afterwards, therefore the Community edition is the optimal choice.

## Git, GitHub and GitHub Desktop

Git [[[11]](#endnote-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.



Figure Git Project Logo

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).

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].

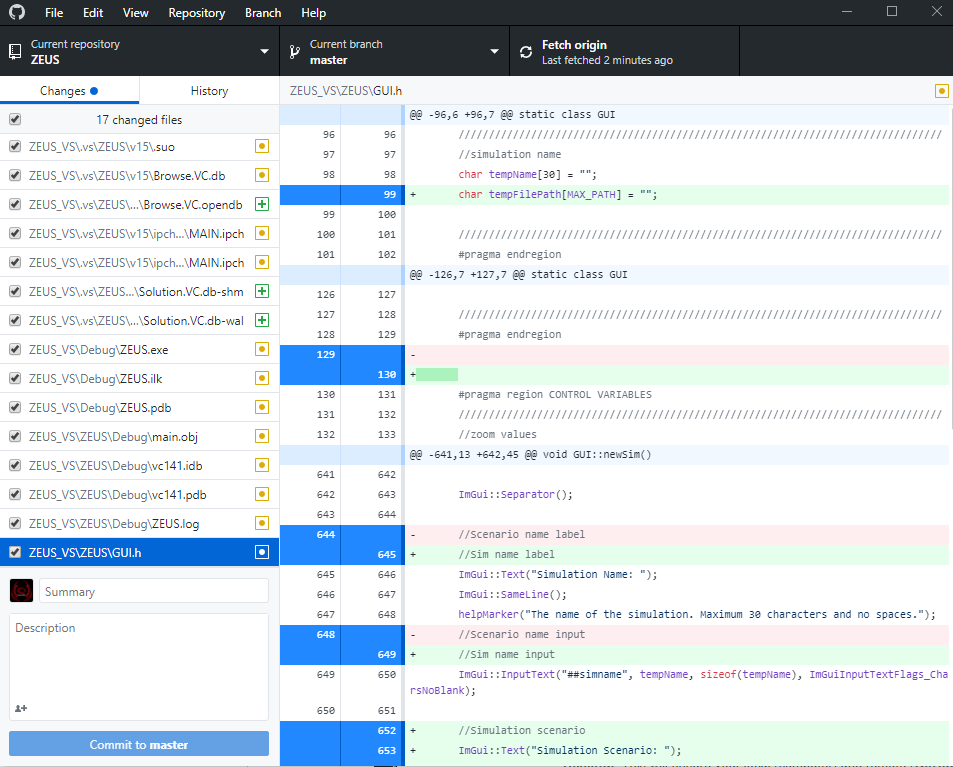


Figure GitHub Desktop UI

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.

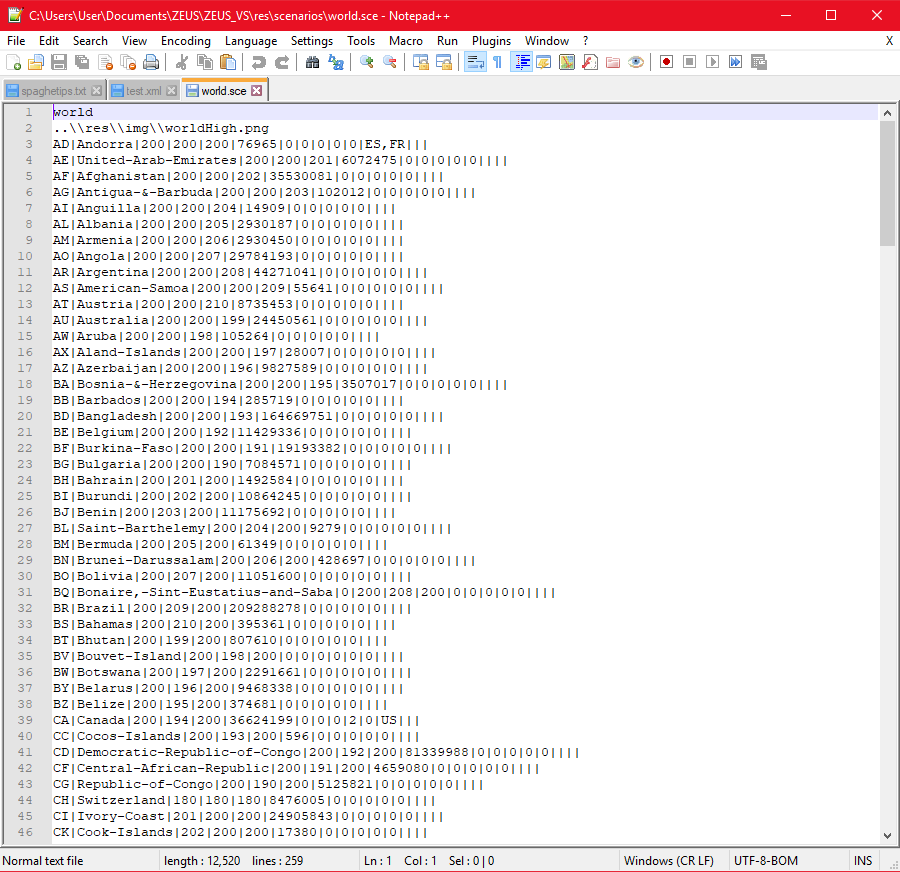


Figure Notepad++ and a scenario file

## Notepad++

Notepad++ [[[12]](#endnote-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.

## 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]](#endnote-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.

### SDL (Simple Direct-Media Layer)



Figure SDL Project logo

SDL [[[14]](#endnote-14)] is a cross-platform library originally created by Sam Lantinga [[[15]](#endnote-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.

#### 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]](#endnote-16)]>)

#### 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.

### ImGUI (Immediate mode GUI)

ImGui also known as dear imgui [[[17]](#endnote-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.

#### 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.

#### 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

# System Development

## 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.

### SDL variables

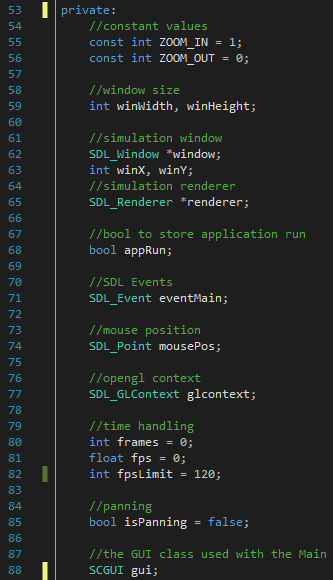


Figure SCMain variables

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.

#### SDL Window

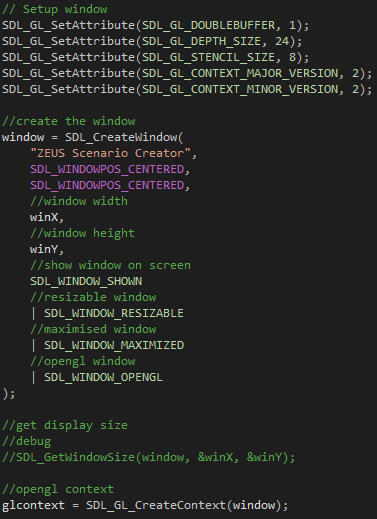


Figure Creating a window

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.

##### SDL Window error checking

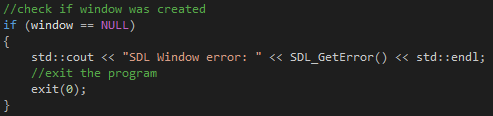


Figure Error checking in SDL Window creation

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.

#### SDL Renderer

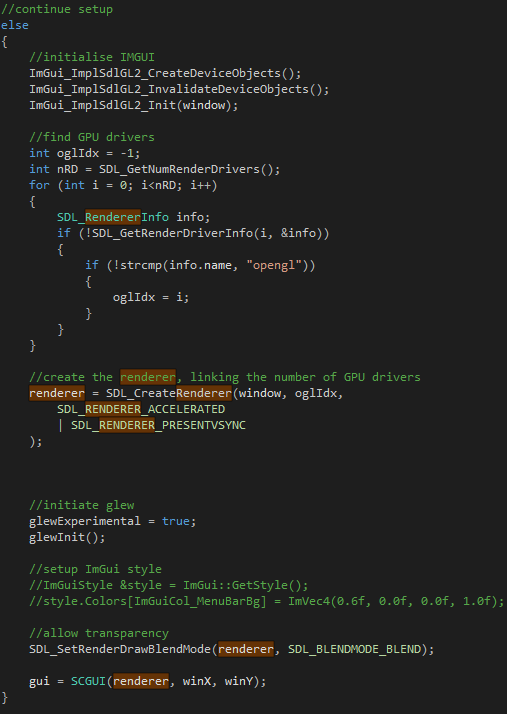


Figure Creating the renderer

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 5.1.1.1. The second parameter is the number of graphics drivers that the renderer is able to utilise; this will vary between machines.



Figure 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.

#### SDL Event

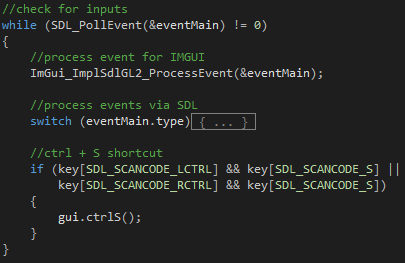


Figure Event Handling in Scenario Creator

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]

##### Switch case



Figure Event for when window is closed

###### 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.



Figure Left and right click events

###### 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]

###### Mouse wheel

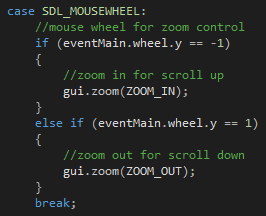


Figure Mouse wheel case

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

###### Single key presses

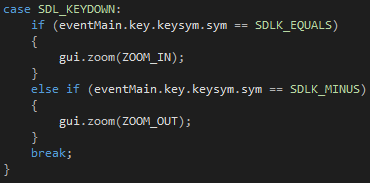


Figure Key Press case

Single key presses act somewhat different to the key presses shown in section 5.1.1.3. 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.

### 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.

#### Setting up frame rate values

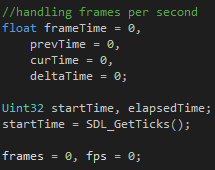


Figure Frame rate variables

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).

#### Code explanation

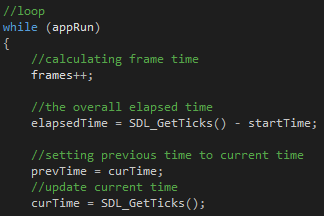


Figure Start of the loop execution

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.

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.

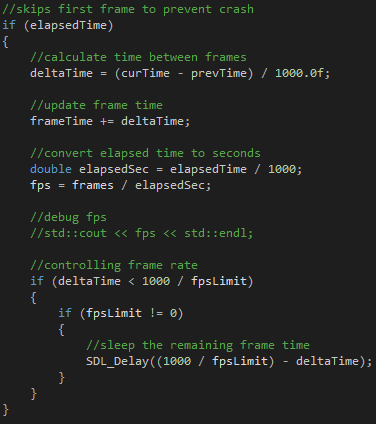


Figure End of loop execution

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.

## Zeus Scenario

### Main.cpp

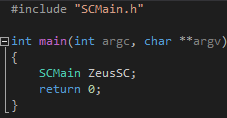


Figure contents of main.cpp

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. 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.

### SCMain.h

### SCGUI.h

## Zeus Main System

### Main.cpp

### Main.h

### GUI.h

## Shared Objects

### Data Handler

#### loadScenario

#### loadSimulation

### Country Objects

### 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.

#### Panning

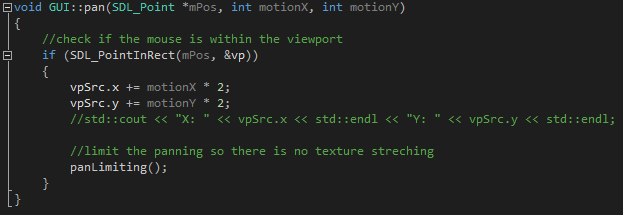


Figure Pan function in GUI.h

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.

##### Pan limiting

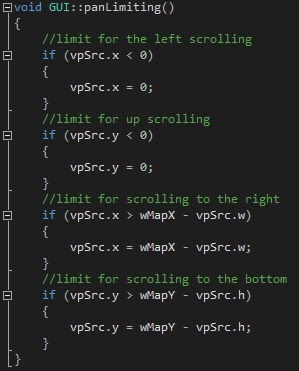


Figure Pan limiting function

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.

#### Zooming

#### Keyboard shortcuts

#### Country name display (on hover)

#### helpMarker

Figure 31 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.

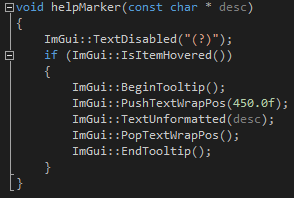


Figure helpMarker function

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.

### Eyedrop tool

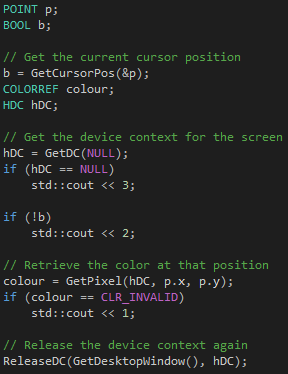


Figure Colour detection

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 I hovering over, making it easier for a user to create a scenario.

Source of the code [[[18]](#endnote-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.

### Default Scenario

### Default Simulation

# System Testing

## ZEUS Scenario Creator

## ZEUS Main System

## Integration Tests

# Usability evaluation

# Project outcomes (lessons learned)

# Conclusions and Evaluation

## Project Conclusion

## Project Reflection

# Future of the project

## Possible Improvements

## Fixing later discovered bugs

# Glossary

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| FPS | Frames Per Second; the number of frames the program renders in a second |
| CPU | Central Processing Unit; component in the physical computer, which carries out program instructions. |
| 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]](#endnote-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 |

# Appendices and 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>

   [↑](#endnote-ref-1)
2. [] ZEUS System GitHub repository [Online] [Available as of April 2018]

   <https://github.com/Czar-Ec/ZEUS>

   [↑](#endnote-ref-2)
3. [] GLEAM Simulator [Online] [Accessed 26 March 2018 09:33]

   <http://www.gleamviz.org/simulator/> [↑](#endnote-ref-3)
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> [↑](#endnote-ref-4)
5. [] Zombietown USA Source (Github page) [Online] [Accessed 26 March 2018 20:43]

   <https://github.com/mattbierbaum/zombies-usa> [↑](#endnote-ref-5)
6. [] Zombietown USA Browser Demo [Online] [Accessed 26 March 2018 20:45]

   <http://mattbierbaum.github.io/zombies-usa/>

   [↑](#endnote-ref-6)
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>

   [↑](#endnote-ref-7)
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> [↑](#endnote-ref-8)
9. [] Visual Studio IDE download page [Online] [Accessed 04 April 2018 16:45]

   <https://www.visualstudio.com/vs/> [↑](#endnote-ref-9)
10. [] Visual Studio Professional and Enterprise edition pricing [Online] [Accessed 04 April 2018 17:00]

    <https://www.visualstudio.com/vs/pricing/> [↑](#endnote-ref-10)
11. [] Git main web site [Online] [Accessed 04 April 2018 18:00]

    <https://git-scm.com/>

    [↑](#endnote-ref-11)
12. [] Notepad++ Webpage [Online] [Accessed 04 April 2018 23:30]

    <https://notepad-plus-plus.org/>

    [↑](#endnote-ref-12)
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>

    [↑](#endnote-ref-13)
14. [] SDL library homepage [Online] [Accessed 06 April 2018 16:46]

    <https://www.libsdl.org/> [↑](#endnote-ref-14)
15. [] Sam Lantinga LinkedIn page [Online] [Accessed 06 April 2018 04:54]

    <https://www.linkedin.com/in/sam-lantinga-02771b10/> [↑](#endnote-ref-15)
16. [] GNU Lesser General Public License [Online] [Accessed 06 April 2018 05:15]

    <https://www.gnu.org/licenses/lgpl-3.0.en.html> [↑](#endnote-ref-16)
17. [] ImGUI (dear imgui) GitHub page [Online] [Accessed 08 April 2018 09:15]

    <https://github.com/ocornut/imgui> [↑](#endnote-ref-17)
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>

    [↑](#endnote-ref-18)
19. [] GPL (General Public License) Web page [Online] [Accessed 04 April 2018 18:20]

    <https://www.gnu.org/licenses/gpl-3.0.en.html>

    [↑](#endnote-ref-19)