Joshua Netherton

16nethertonj@student.cleeveschool.net

0154

H446/04 Programming project: Nuclear Reactor meltdown Simulator

Joshua Netherton

Contents

[Analysis 2](#_Toc118722702)

[1. Defining the problem and the Stakeholders 2](#_Toc118722703)

[Who are the Stakeholders? 2](#_Toc118722704)

[What do the Stakeholders do? 2](#_Toc118722705)

[What problem do they have? 3](#_Toc118722706)

[How will they make sure of your proposed solution and why is it appropriate to their needs? 3](#_Toc118722707)

[2. Justification of how the problem can be solved by computational methods 3](#_Toc118722708)

[Thinking abstractly and visualisation 3](#_Toc118722709)

[Thinking ahead 4](#_Toc118722710)

[Thinking procedurally and decomposition 4](#_Toc118722711)

[Thinking logically 5](#_Toc118722712)

[Thinking concurrently 5](#_Toc118722713)

[3. Research 5](#_Toc118722714)

[Initial Research 5](#_Toc118722715)

[Interview 9](#_Toc118722716)

[Deliberation 10](#_Toc118722717)

[Features of your proposed solution 10](#_Toc118722718)

[Hardware & Software Requirements 11](#_Toc118722719)

[Success Criteria 11](#_Toc118722720)

[Design 14](#_Toc118722721)

[1. Top-down diagram 15](#_Toc118722722)

[2. Systems breakdown 15](#_Toc118722723)

[Proposed Program Design 16](#_Toc118722724)

[Revised Program Designs 19](#_Toc118722725)

[Algorithms 21](#_Toc118722726)

[Key Variables 30](#_Toc118722727)

[3. Test Data 32](#_Toc118722728)

[4. Test Table Plan 32](#_Toc118722729)

[5. Summary 35](#_Toc118722730)

[Developing the Coded Solution 36](#_Toc118722731)

[1. Prototype 1 36](#_Toc118722732)

[Written Code 36](#_Toc118722733)

[Testing 39](#_Toc118722734)

[Plans for Prototype 2 46](#_Toc118722735)

[2. Prototype 2 46](#_Toc118722736)

[Algorithms 47](#_Toc118722737)

[Written Code 49](#_Toc118722738)

[Testing 50](#_Toc118722739)

[Plans for Prototype 3 57](#_Toc118722740)

[Table of Figures 58](#_Toc118722741)

[Bibliography 59](#_Toc118722742)

# Analysis

Max (10 Marks)

## Defining the problem and the Stakeholders

### Who are the Stakeholders?

My Stakeholders are individuals training in nuclear technology and energy. Designers and engineers would use this program to test and look at customisable preconditions and see how they would unfold in real world weather and locations. They will then be able to practice regaining control over a dire situation and simulate how their decisions impact the results of a meltdown. This instant feedback in a safe environment is what will allow them to better train prior to getting onto the job and increase security in their field.

It will help to give them insight into how best to think about where new plants should be built, and what it is that will affect the outcomes of different scenarios. It can also be looked at to measure how long certain events will take to cause serious damage in different conditions and locations, which can be used to test response teams’ timings and ability to act before large amounts of damage is done. Following on from this, it will allow them to see how they (the user) manages themselves when under stress, along with how their decision-making skills change and what impact that has.

### What do the Stakeholders do?

The Stakeholders this program has been designed for are going to be individuals in the field of nuclear energy. Some may be in training to work in a plant, and this program will allow this stakeholder to investigate not only the consequences of something going wrong but also how they can best resolve a dire situation. My Stakeholders could also be in the design industry of nuclear power plants and could use this program to look at where best to build a plant by looking at how bad the damage of common issues with nuclear plants would be.

My Stakeholders could also be university students training to into the nuclear sector or are already in training to become a plant operator. They will be learning about the effects a nuclear reactor meltdown could occur, such as contamination, radiation sickness, irradiation (Morgan and Sowa, 2015). They will be focused on researching development and outcomes of a nuclear powerplant and will still be in the learning phase. They may also be studying nuclear power, and can use my program as a research tool to test hypothesise they have come up with.

The worst possible outcome while working at a nuclear plant is any aspect of its running being in a state which the operators cannot control. My Stakeholders will be training to operate and work with reactors and will learn about the possible outcomes of losing control. If the core is allowed to overheat beyond its tolerances, then it will meltdown. China Syndrome is a *“hypothetical sequence of events following the meltdown of a nuclear reactor, in which the core melts through its containment structure and deep into the earth.”* (*China syndrome*, no date), this is a worst-case example of what a meltdown can lead to and something my Stakeholders may train for. While a China Syndrome has never actually occurred, there are lesser eventualities that do, and these are things my Stakeholders will need to learn about and prepare for.

### What problem do they have?

The problem they have comes from the lack of functioning, public simulators such as this one, which people can look at and experiment with. Following my own research there are little to no meltdown simulators on the web that have the same specification as my own. While nuclear power plant simulators alone do exist – while many seem complicated to use – there are no easily accessible meltdown simulators, which is what I intend to create.

The IAEA (International Atomic Energy Agency) does provide some useful simulators that are very specific to certain environments (*IAEA (no date)* *Nuclear reactor simulators for education and training*, *Iaea.org*.) however, many forms need filling in and permission from government organisations is required to access them. This does not provide widespread accessibility to anyone wanting to learn, a problem my program will solve.

While my Stakeholders may have these details as some will work for governmental organisations such as EDF (*Homepage-July16*, no date) or Dominion Energy (*Home*, no date), they do not have the ease of access I intend my program to have. My Stakeholders may also extend to average citizens, who will not have these sorts of details and in turn will not be able to access these resources.

### How will they make sure of your proposed solution and why is it appropriate to their needs?

The simulator will be very interactive, allowing the user to control a multitude of variables which would contribute to the damage (or lack thereof) caused by the meltdown of a nuclear reactor in varying conditions. This unique control will mean real world events can be acted out from inside my program and will have a fidelity that not many free services have.

It will allow them to test, experiment, and become more climatized to the possible damage the reactors that they are working on could cause should something go wrong. Allowing them to act out real world scenarios on a computer. This increases awareness will help to increase caution and reduce risk, allowing designers and engineers alike to experiment within a safe environment.

It will also allow users to attempt to regain control of a melting down reactor and what steps can be taken to reduce damages as much as possible. When in a learning environment it is easy to forget the magnitude of the situation you are in, and the information you are being given becomes numbers across a page. If you are able to control the initial meltdown along with the steps you need to take to restabilize a reactor then more experience will be gained and better recollection of the ideas you’ve learnt will stick.

## Justification of how the problem can be solved by computational methods

### Thinking abstractly and visualisation

My simulator will be designed to show the outcome of certain conditions by a radius of damage around the reactor, and how certain condition and environments affect the outcome of a reactor meltdown in a location decided by the user on an interactive world map. This is simplified from reality by the fact that none of the day to day running of the plant will be a variable, nor factors such as age or construction quality. The focus will be on a standard model using standard problems and outcomes.

It will be designed primarily with my Stakeholders in mind, who already have a good understanding of how a plant would work and are looking to feed their curiosity on what sort of damage the worst-case scenario could cause. With this in mind, I will not be describing how different aspects of a nuclear powerplant work, as this will be unnecessary detail for the users of my program.

Following the setup of the meltdown, the stakeholder will then be able to control different aspects of the reactor to cool the core and reduce the impacts of the meltdown the user has setup. This will give the user more options for what they do but I will ensure that my program does not become overcrowded with options and complicated to use. Only crucial variables will be controllable, and they will be controlled via a sliding bar, a serious abstraction from the real world but still providing enough fidelity for the user to make an impact with their decisions.

### Thinking ahead

For my program to successfully run only a limited amount of data will be needed from the user themselves: a location of the meltdown which will be given via an interactive map; the date in which they would like this meltdown to occur – this allows for that days weather data to be included -; they have the choice to either put the program into online or offline mode; and the severity of the meltdown (how bad it is).

Once these inputs have been given the program will simulate the disaster, to do this there is data needed to output an accurate result – given the program is in online mode (if it is offline then any impactful data will be unnecessary). Data required includes weather data to show how radiation might travel; the result of the different severity meltdowns – specifically the one the user has chosen so that the output can be made worse/better; and how this respective meltdown could occur so that a short blurb of information can be given to the user.

Now that the meltdown has happened, and the user has been shown the effects of the situation, they will move onto controlling various aspects of a reactor, such as steam flow, water flow, control rods depth, etc…. The inputs the user will provide will impact how the meltdown continues to unfold and the challenge will be to bring the reactor back down to acceptable temperature methods and avoid a full core meltdown. User inputs will be given via a slider on screen for the different options, and the program will provide dynamic feedback.

### Thinking procedurally and decomposition

My problem can easily be broken down into smaller chunks and tackled like that. I have produced a small checklist to breakdown my project into the separate parts that need solving. I will be creating a top-down diagram in the Design section of my program along with flowcharts outlining different functions and procedures. This will allow me to break down my program into the most basic aspects it can come in and focus on one small thing at a time. The idea of these diagrams is that I could give it to a professional programmer, and they could produce a program which does exactly what I picture it doing, without having to ask any further questions.

When coding my project, I will be dividing it up into sub-procedures and different functions. This not only helps when it comes to readability, but I’ll be able to compare the functions I have inside my program with my detailed success criteria to ensure that I have every box ticked along with the pre-produced flowcharts. This also allows for easier reusability if sections of code need repeating.

Breaking my project down allows me to focus on one thing at a time, this keeps things simpler and when I get stuck on one problem I can take a break, I can go elsewhere and work on something else and return to it later. Decomposing my program means I don’t need to spend my time reading through lines and lines of code, worried I’ll break one thing while working on another, which overall will increase the efficiency of my programming. It also allows for the use of local and global variables, and the more I can use local variables in functions and procedures in my program, the better the space efficiency will be, further increasing accessibility for my users.

### Thinking logically

My program will have obvious decision points where branching or looping could take place. In situations where weather data isn’t available then the program will just skip any code regarding simulating outcomes with weather data and will just output an average result. If there is that data available, then it will use it. Taking this a step back if the program has been put into offline mode it won’t even check for available weather data and will just move on.

Looping can take place when looking for any towns/cities within the radius of the blast zone, a loop checking to see if any points in the blast zone contain housing (a value I hope to get from the maps API) would be a good example of this. Iteration can be used here to set a condition causing the program to keep checking for a certain item that matches certain targets.

This allows my program to exhaust all possible options in a logical manner and removes the possibility of something being missed, this will further the useability of my program as more parts of it are automated. The more decisions that need to be made also add to the fidelity of my program and add to a more realistic program. The hope is to have it run in such a way so that when put into online mode no two meltdown simulations made on different days are identical as naturally the weather is always subject to change.

### Thinking concurrently

A good example of how this program will run concurrently is I will have the UI running at the same time as the bulk of the simulation/back-end code and the code that processes the user inputs. These will have to run simultaneously or else the UI would close the moment the user inputs any data.

I will also need to have the topography checker running alongside the weather processer to ensure that things don’t happen one at a time and the user is left waiting a long time for their result. Once the user initiates the meltdown of the reactor all these parts will need to occur simultaneously therefore my program will need to be able to run concurrently.

Once the user starts to control the aspects of the reactor, the data provided will need to be processed concurrently to allow for live feedback to the user, so that they can know what the impacts of their actions is. Should the program need to run everything one after the other, the results of a change the user made wouldn’t be noticeable until time later, rendering it useless.

If this were not to occur then it would do one whole job after the other, and the user would also be sat waiting for a very long time and would quickly get bored. While things are processing in my program, I also intend on having a moving loading symbol so that the user can see the program has not frozen, this will need to run side by side with all the display I have, along with the simulation calculations.

## Research

### Initial Research

Following some research, I have found little to no freely accessible reactor meltdown simulators. There is an abundance of nuclear missile radius websites however the meltdown of a nuclear reactor does not seem to be a hot topic. I did manage to find a couple power plant simulators, where you can control the plant itself. These provide some insight into what I would like the game to look like, however, these do not accept many inputs and are primarily just to look at how a plant works.

As you can see in the image below taken from the Nuclear Institutes website depicting a nuclear reactor, much of it is designed to be educational – however. My program will take more inputs and give more outputs to the user to educate them on the damage that a meltdown could potentially cause if the worst should occur. While this website is good, it is well outside my project’s scope.

A screenshot of a computer

Description automatically generated with low confidenceCurrently this is the top website that pops up when a user Googles Nuclear Reactor Simulator. While holding a lot of educational value I do not feel it provides a simulation experience and does not allow any form of user interaction other than the selection of what the user wants to learn more about. My program will be far more focused on giving an immersive experience and teaching the user what can happen as a result of something else.

Figure - Nuclear Reactor Simulator (no date) Nuclearinst.com

#### Nuclear Institute Nuclear Reactor Simulator (Nuclear Reactor Simulator (no date) Nuclearinst.com)

The Nuclear Institute has created an info guide Nuclear Reactor page, it is very visual and has been built to help beginners learn about how a Nuclear powerplant works. It is fully functional via a web browser and has provides interactive insight into how everything works. I have enjoyed the time using it so far and have come across little problem with usage. A screenshot of a computer

Description automatically generated with medium confidenceAs we can see here it provides a close-up view of the control rods along with what they do inside of a nuclear reactor. It has this functionality for every aspect of a reactors functionality and allows you to explore.

Figure - Control Rods of Nuclear Reactor Simulator (no date) Nuclearinst.com

The Nuclear Institutes decision to use graphics for this aligns well with what I assume to be their goal to teach users about how a nuclear reactor works. Using graphics and moving images such as these is a good way to demonstrate to a user, along with the text visible on the right-hand side, this program has clearly been designed to teach its user how a reactor works. While this idea is good, it may not be something I do too. My Stakeholders will primarily be people who already have a good understanding of how a nuclear reactor works and will primarily want to focus on the damage aspect. This 3D rendering could become the focus of my program – taking away from what the foundation of what I am trying to create. It would also increase the hardware demand of my program, something I am trying to limit.

##### Advantages

It provides a good explanation of how things work and seems to not demand too much processing power to run. It is partially animated and gives information of how different parts work. This provides the user with an easy-to-follow learning experience, which really allows you to get into the grit of how a plant functions on a very high level. This would be an excellent teaching resource for many age groups and would be a good resource for anyone doing a project on reactors, allowing the user to look at exactly what is going on.

##### Disadvantages

It does not provide any information regarding a meltdown of this reactor and is not built with my stakeholders in mind. The people using my program will primarily already have pretty good knowledge of how a reactor works and will not need the detail that this website goes into. It has clearly been designed to show people how a reactor works, and not the outcome of a meltdown.

It also has some performance issues, wherein the movement is rather jittery when you zoom in on different sections of the plant. This can hinder the user experience as it becomes tough to look at. The full screen mode also does not work at all, the interactive section stays the same size and the rest of your screen is filled in blue.

#### NUKEMAP by Alex Wellerstein (NUKEMAP by Alex Wellerstein (no date) Nuclearsecrecy.com)

NUKEMAP by Alex Wellerstein (*NUKEMAP by Alex Wellerstein* (no date) *Nuclearsecrecy.com)* is the closest website I have been able to find which represents what I want my programs UI to look and feel like. It is built to show the radius of damage done by launching a customisable nuclear bomb at anywhere in the world. (As shown in Figure 4).

Alex Wellerstein’s decision to have a 2D map and a moveable marker comes from the program’s primary focus to just output the damage that different nuclear bombs would do to any location on the globe. The information given to the user on this site is strictly just about the damage done and contains no information on how the bomb itself works. 3D rendering or information on how a nuclear bomb has been abstracted from the user, and our focus is directed to what the bomb does.

We can assume Alex Wellerstein has done this to not only keep his program simple, but also create an online resource anyone can use for the purpose of finding out what damage a nuclear bomb would do, and nothing else.

Figure - NUKEMAP by Alex Wellerstein (no date) Nuclearsecrecy.com

##### Advantages

This shows the UI baseplate of how I want my program to look. It has a very well-built navigable map which is precisely what I want my program to have too and shows the radius of damage in a similar way to how I indent on doing it. It also allows you to select the type of nuclear bomb you wish to detonate – now naturally I will not be using nuclear bombs, but I want to do something similar with different types of nuclear reactor/different sizes.

##### Disadvantages

The most obvious point being this is for nuclear bombs, not a meltdown. It also does not go into the same amount of detail in terms of damage done that I wish to go into. I’m looking to output a highly accurate result which would represent damage done if a meltdown occurred. The UI is very wordy too, which is something I’d like to avoid.

It also does not have any further control scenarios, which naturally would be harder with a nuclear bomb, but my program will go onto allow the user to try and reduce the damages caused by the meltdown as much as possible.

#### Nuclear Power Plant Simulator (MH (no date) Nuclear Power Plant Simulator free online game, Github.io)

This website is designed to simulate you as the controller of a nuclear power plant. It permits a lot of control over how a plant would work and if you are not able to supply a dynamic energy demand with power from your plant, you may end up failing. You can also cause a meltdown which is also a failure. This website struggled to load on the device I used but the essence is there is a lot to manage and control which may be enjoyable for someone with more knowledge on how a plant works.

Figure - MH (no date) Nuclear Power Plant Simulator free online game, Github.io.

The creator of this program has used an interactive base to create an immersive experience for the user. This decision has most probably been made with the focus of the program to be controlling a nuclear reactor. At its essence this program is a game which the user can play.

Naturally outputs are a feature of this website, things like power output or temperatures are clearly visible in the photo above, but these are an aspect of the game, and are defined by user decisions. This has been done to create emersion for the user to simulate the feeling of controlling a nuclear plant. The outcome of an unsuccessful run is a nuclear meltdown, and we can assume this has been done to create a sense of risk/reward for the user. Do they risk upping power output to meet demand but potentially overheat the reactor? Or do they maintain safe temperatures but risk not being able to keep up with energy demand.

##### Advantages

This platform is clearly designed to be used by people with more knowledge and understanding of how a power plant would work. This is like the scope of my project as the stakeholders I am directing mine for are people who understand how a plant would work. It is very high fidelity and could be considered quite an immersive experience. The task of managing power output while trying to prevent the reactor from overheating gives the user fast paced gameplay where quick, decisive decisions need to be made.

##### Disadvantages

This program seems to take place in the time period before that of my program. In this simulator you are providing power to a population and with this need to manage temperature and a multitude of other variables to ensure you do not overheat the program and cause a meltdown. My program however will allow the user to cause a meltdown and then tasks them with mitigating the damage caused, and so the scope of these programs is different in this respect.

It also provides more options than my program will, even once the user has moved onto the recovery stage. It is much less abstracted than my program will be, and by extension increases the difference in scope.

The website also feels cluttered and overly busy, along with seeming to have loading issues with certain objects. This will reduce enjoyment of the user experience and may result in users leaving. This is something I will try to avoid when producing my program. All the objects will be preloaded into my program, as I intend for it to be able to run fully without internet access.

### Interview

Q: *What would you want the platform to feel like?*   
A: I want it to be easy to navigate and control. A smooth interface is crucial, but some programs take this step to far and their product begins to be clunky and seems unresponsive. A well-polished map built into the surrounding input AI would be perfect, and something that doesn’t require tons of processing power.

Q: *Do you want descriptions on how the reactor functions?*  A: I think it may be useful to have this. While the people like myself may already have this understanding, users not as well versed may find this useful. Do not go into too much detail as this could contribute to the program looking messy and cluttered. Just a brief overview of what happens and why meltdowns occur would be good.

Q: *Would you like different types of reactors to choose from? Or rather simply one reactor and you can drag and drop wherever you choose.*  A: I think this would be a cool feature but there’s no need to cover every single type. I think sizes are more important, for example a PWR (Pressurized Water Reactor - *Linquip Team. (2021) All types of nuclear reactors (PDF & Charts), Industrial Manufacturing* Blog *| linquip)* used in naval propulsion melting down will result in a much lesser damage zone compared to theChernobyl LWGR (Light Weight Graphite-Moderated Reactor - *Linquip Team. (2021) All types of nuclear reactors (PDF & Charts), Industrial Manufacturing* Blog *| linquip).*

Q: *Should there be an option to just view a basic damage radius? (Similar to NUKEMAP by Alex Wellerstein)* A: Yes definitely, if by that you mean the ability to see a damage zone without it using windspeeds/topography, and it just show a radius of damage, definitely I do.

Q: *Would you rather this be a web accessible service or something you can download?* A: Having this as either would work, naturally it being a web accessible program means you need a connection to use it, but this would apply either way as it would need a connection to use real time weather date etc... but if you include an option to just view damage zone, then a downloadable one would work completely without internet, so this may be the best bet.

Q: *Do you want this program to be able to view the placement of location in 3D? (At street level)* A: In my opinion this would be a cosmetic upgrade, something that is cool but completely unnecessary. If it’s something you could do, do it. Don’t spend all your time fixated on it, however.

Q: *Would you like to be able to input custom conditions such as wind and climate*? A: Similar to the question above, I think if there is the option to not use real time weather data i.e., it has an offline mode, I don’t think this feature would be critical, but could be cool to have.

### Deliberation

Following a conversation with an example stakeholder I have walked away with an abundance of useful information on how my project should look once it is completed. I will now be focusing more on the UI look and feel, making sure it is smooth and well-polished, to maximise user enjoyment.

I will most probably be doing this as some sort of downloadable program to reduce the need for internet access to use this, along with including an offline option for the user to interact with.

I won’t be focused too much on information being given to the user about the reactor as my stakeholders are primarily people who understand how they work well. Along with not looking too far into 3D rendering of my program, but this may be something I investigate once the main program has been made.

To conclude, my main focuses will be on a friendly UI experience, along with ensuring the key aspects of my program work well, like the navigable map and weather data affecting the outputted damage radius. Only once this is done will I consider looking into the more complex ideas I have come up with. The foundation is important, and the focus will be getting this built well, only once this is the case will I allow myself to focus more on cosmetic upgrades of the program.

### Features of your proposed solution

Once I have reached an end point for this project, I hope to have a functioning map you can interact with and place down a nuclear power plant at any place in the globe. Then the user should be able to initiate a meltdown and using real world topology and weather data, the program will output the area affected by this meltdown (Similar to NUKEMAP - *NUKEMAP by Alex Wellerstein*) which will be dynamic depending on the conditions the meltdown occurred in.

Once this has happened the user will then be able to control various aspects of the plant using sliders and a simple heads-up display to see the results of their actions, to attempt to stop the meltdown and save the reactor. A display showing the radius of damage will also be visible, which will change depending on how well the user is doing.

Following conversations with an example stakeholder I will be designing this program as something which you could download. Realistically, this will not be something that you can run without loading up the code itself and running that due to limitations in both my knowledge, skill, and time in which I must complete this project. In other instances, I may also consider implementing an online version, but again due to time and knowledge constraints, this is something I will not be doing.

As of right now I am unsure whether the dynamic weather and topology of the local environment will be fully implemented by the end. If I am able to find a good weather data API which I can use, I imagine I should be able to do the former, but if not, then this may be something I am unable to do. In terms of including topology, this would be a massive aspect of the project as the entire globe would need to be mapped, along with a database storing how different geographical features would affect a meltdown, and simply there may not be the time to do this.

I do however hope to include a feature that means if there is any sort of settlement/town within the damage radius a message will pop up informing the user that this is somewhere that would need to be evacuated. My initial thoughts are that this is something well within the realms of possibility and is something I would like to include within my final program.

### Hardware & Software Requirements

To begin, if my program is to have any sort of real time data included then internet access will be a bare minimum. I do plan on including an offline mode of the program allowing a user to just view a basic circle around the reactor which displays the different levels of damage that would occur, but to view anything to do with real world weather or similar then an internet connection will be required.

Following conversations with a prospective stakeholder I have decided that there won’t be 3D rendering of a model reactor, this isn’t necessary for my program in the slightest and would only ever be a purely cosmetic feature I would add in the future should this not have been a A-Level project. This does reduce the hardware requirements of my program dramatically, it will not need a powerful processor, buckets of RAM, or a beefy GPU. With the average 8GB of DDR4 (or above) RAM, an Intel Pentium (or equivalent) and above and integrated graphics or above should cover all my bases well.

These decisions have been made so that I can ensure that nearly any modern equipment (tech from the past 3-6 years) can run my program with little to no difficulty. My success criteria requires that aspects of my program can run with little to no delay or lag. This will not require an abundance of processing power to achieve, as I will not be rendering anything, only performing calculations and outputting their results. Hence why my lesser hardware requirements are all that are needed.

In terms of software required, naturally I will need additional software to program with, such as Visual Studio code (*Visual Studio Code - code editing. Redefined*, no date). While basic IDEs do the job, I will be swapping between languages for various parts of my code and feel I would benefit from automatic syntax completion and the more advanced debugging that something like Visual Studio would offer.

While using Visual Studio Code (*Visual Studio Code - code editing. Redefined*, no date) I will also be importing libraries and using built in extensions like code colour coding, automatic syntax completion and Visual Studio extensions designed to maximise user efficiency and ease of use. An important part of coding is ensuring you understand what it is you are programming, so auto completion of code will be avoided, despite there being extensions able to do this. Any code I use from the internet will be properly accredited.

### Success Criteria

|  |  |
| --- | --- |
| Criteria for success | Completed? |
| The program runs in a fully independent window from the moment the code is ran. |  |
| An interactive map works within the window. |  |
| The map has placenames which are up to date with real world location data. |  |
| You can navigate this map with ease and do not have to struggle with accidentally interacting with the window, i.e., scrolling on the map doesn’t scroll the window. |  |
| The map is on the left and has other program controls to the right. (Outputs on one side, inputs on the other) |  |
| The surrounding space isn’t cluttered with words and has a clean appearance. The user should not feel overwhelmed by excessive word counts on the main page. |  |
| There is a dropdown menu on the page which allows you to select the type of meltdown you wish to occur. This appears as a list which you can scroll on and orders the different type of meltdowns in a best to worst order. |  |
| Once one has been selected then it auto fills the previously blank box you clicked to view the dropdown menu with the meltdown scenario you selected. |  |
| On the map you are able to easily select a location with a drop marker, which may be located in the middle – maybe like a crosshair – and you drag the map to line up with this position. |  |
| The location displayed is visible somewhere on the screen to confirm with the user that they have their desired location selected. |  |
| There is a box where the user can fill in a date. This will appear as a calendar on a monthly view. You can click the month at the top of the calendar to see all the months in a year, and then the year at the top of this page to see all the years – to allow for quick and easy navigation of various dates. |  |
| This calendar will only allow you to select dates which have weather data available, too far back or too far forwards will not have recorded weather data and therefore won’t be able to give my program any information. |  |
| The box you click to choose the day will automatically select today’s date unless the user edits this. |  |
| There is a detonate button below/near to where you select the type of reactor which is obvious and easy to click. Once you have selected a type of meltdown you can click this button and it will output a destruction radius around where your marker is placed. You should be able to move the map and the radius will stay in place once the detonate button has been selected. |  |
| If no meltdown type has been selected an error message will appear informing the user that they need to choose a type of meltdown. This won’t need to happen for the date as it will assume today’s day if nothing is selected. |  |
| A dropdown menu which doesn’t appear different in terms of colour relative to the background is accessible called “Read More”. Selecting this will display more information about the meltdown type selected. |  |
| Information about user decisions should be stored by the program so that it can access these later. It should also store geographical location selection so it can look at popular places which people like to frequently look at the meltdown damage of. |  |
| This data can be used to change where the marker is initially placed on the map upon launch of the program. If there is somewhere many people like to look at – such as New York – I may have it so that upon launching the program the location marker is already placed in New York. |  |
| If internet access is not available when the detonate button is selected, given that the program is in online mode, then the program will output an error to the user. This error will include the option to put the application into online mode, allowing the user to continue using it but without real time weather info. |  |
| A switch button will be visible on the main page, this will be labelled offline mode on one side, online mode on the other (left and right respectively). Clicking this switch will allow the user to toggle the application between offline and online modes depending on what they wish to see. |  |
| Once the detonate button has been selected if there is a loading time – i.e., any time where the program isn’t outputting something – then a moving object (loading circle, loading bar etc…) will appear with the purpose of showing the user that the application hasn’t crashed. |  |
| Once all fields have been filled in correctly, and the program is done loading a radius of damage should be outputted on the map. This radius will consist of different sections depending on what damage will be done. (Red in the middle for actual blast, a yellow much further out indicating places which could be affected by contamination). |  |
| A key is visible somewhere on the application, which will indicate what the different colours in the damage radius indicate. These shouldn’t be overly wordy and will exist to provide a brief overview of what they mean for the user. |  |
| If weather data is accessible and there is internet connection, the output radius should be correctly manipulated by things such as wind or rain. I will be able to check this by looking at real world event and comparing that to what my simulator outputs. |  |
| Topology should also impact the effect of my output radius. Mountains will decrease the distance the damage can reach, whereas flat land will increase it. |  |
| Nothing explicit should be outputted about topology, it will primarily work in the background to provide extra detail to the output radius of damage. |  |
| Any settlements found within the damage radius should be logged by the program. |  |
| If there are places within the damage radius, if the damage caused is bad enough then the program will output a list of the places which need evacuating. |  |
| Once the initial output has happened a loading page shall appear. |  |
| A moving scroll wheel will appear – so the user knows that the program has not frozen. |  |
| The page will then change to an interactive screen with sliders corresponding to different controls over the reactor. |  |
| The map will still be visible on the right. |  |
| Ensuring that all user inputs are on one side of the page and all program outputs are found on the other side. This allows for a friendlier user experience. |  |
| Each control will be of equal size, with its name being above the slider. |  |
| A small description of its use will be below the title, only a basic use case description. |  |
| Values being set will be visible, such as water flow, the movement of the slider will change how much water is flowing, and then output this value to the user. |  |
| Next to each control a red, amber, or green light will be visible, this will represent whether the user is making things worse, not making any difference, or making things better respectively. |  |
| There will be some gauge or scale on this page showing how close to losing control the user is, showing how close the meltdown is to hitting critical mass and no longer being saveable. |  |
| If the user reaches the end of this scale – the reactor has gotten too hot and can no longer be brought back under control – then a window will pop up telling the user, they have failed. |  |
| An option to try again will be visible. |  |
| An option to go back to the home screen will be visible. |  |
| Both will work. |  |
| Should the user be able to bring the reactor back under control and manageable temperatures are reached that can be kept sustainably, then a screen saying success will appear. |  |
| An option saying play again will be visible. |  |
| An option to go back to the home screen will be visible. |  |
| Both will work. |  |
| On both the Failure and Success screen a timer will be visible showing the user how long they played for – this timer will begin once the user has clicked detonate on the home page and the loading has finished. |  |

# Design

Max (15 Marks)

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Figure – Top-Down Diagram (Figma, Whiteboard and Diagram Maker)

## Top-down diagram

Above I have produced a decomposition diagram or a Top-Down diagram. These serve the purpose of breaking down my larger problem into lots of smaller, more manageable chunks. As you can see, I have first broken the simulator down into the three main sections, user inputs, program inputs, and then final output. Then from there I have broken these down into smaller chunks, which I have then got into detail on what I need to do to achieve this part.

I have done this so that I am better able to visualise my program in a light of exactly what needs doing. Leaving this as just “make a Nuclear Reactor Meltdown Simulator” is daunting and complicated. I wouldn’t know where or how to start the project and would frequently need to revisit sections due to missing out aspects when I first did it.

Now I have broken it down I can look at what exactly needs doing and can begin to think about the how. This top-down diagram is also forming the foundation of the functions and procedures I will have in my final program. This allows me to section my program into different sparts and focus on one bit at a time, increasing productivity as any research I do will be focused on one area, and if I’m doing similar sections of the program reusable code will appear more often, allowing me to decrease the amount of debugging I need to do.

## Systems breakdown

### A picture containing text, device, meter Description automatically generatedA picture containing text, device, meter Description automatically generatedProposed Program Design

Figure - Main Program Screen (Flowchart maker & online diagram software, no date)

Figure – Control Screen (Flowchart maker & online diagram software, no date)

A picture containing text, device, meter

Description automatically generatedA picture containing chart

Description automatically generated

Figure – Success Screen (Flowchart maker & online diagram software, no date)

Figure - Failure Screen (Flowchart maker & online diagram software, no date)

#### Main Program Screen

My main program design (Figure 6) has been produced as a wireframe diagram. It includes all the aspects that the user will be able to see and use upon launching the program along with the inputs they will be able to put in. The top right of this page has the option to switch offline mode on and off, giving the user control over whether the program will use weather data or not when calculating the damage radius.

You can also see the map; this is where the user will be able to select the location for their meltdown. They will be able to manually control the map via click and drag but also have the option to use the buttons on the bottom right to control the map if they should choose.

#### Control Screen

Once the user initiates the meltdown, they will be taken to the control screen (Figure 7) which is where they are able to control different aspects of a reactor and hopefully bring the situation back under control. They will have access to sliders allowing them to adjust aspects such as control rods, steam rates, cooling, and other aspects of the reactor that influence the damage it would do should it meltdown uncontrollably,

This will be the page where the majority of information will be given to the user. While user input is required on the left hand of the screen, there is also ample space on the right where information will be given to the user to provide more insight into the environment that the nuclear reactor is in and how their inputs are changing the state the reactor is in.

#### Success Screen

This screen (Figure 8) will only be visible if the user successfully brings the reactor under control. Should they be able to bring the core back down to acceptable levels and fully avoid any possible damages of the situation the result will be deemed a success and the user will be told.

There will then be the option to return to the home page (Figure 6) to restart.

#### Failure Screen

This screen (Figure 9) will only be visible if the user is unable to bring the reactor back under control and the core is allowed to get too hot. This will result in a full meltdown and the result will be deemed as unsuccessful.

The Failure notice will display what went wrong with the reactor and give the user the option to either return home and choose their settings again (Figure 6) or try this scenario again (Figure 7).

#### Justification

I have chosen to make these wireframe designs to help better visualise what my end product will be. My stakeholders will want an easy-to-use interface that doesn’t take time to figure out. Producing early sketches of what this interface will look like can help to pinpoint the areas that may need change/require improvement, along with what areas work. Along with this is also allows me to picture how my program will flow from page to page, and what will happen at each step.

It also gives me something to work from once I begin the coding section. If I went into programming with no picture of how to implement the different sections of my success criteria and top-down diagram, I may end up producing something complex and tricky to use in an attempt to get the features working, without any consideration into what these features will look like together. With this wireframe diagram, I am now able to visualise my ideas better and can work to code the different sections of the design, rather than design different sections of the code I’ve written.

Throughout the design of my program, I have ensured that any decision I have made had the interests of the user as the main focus. On all of the parts to my program I have laid things out so that all the user inputs are on one side and all the program outputs are on the other. This allows the user to direct their focus to one section at a time and not have to constantly flick between thinking about what they’re doing and what they’re being told.

Gestalt Principles (*Gestalt Principles (no date) interaction-design.org.)* are “laws of human perception that describe how humans’ group similar elements, recognise patterns and simplify complex images when we perceive objects.” This is a theory that designers use on websites and other applications to organise content to allow for a more user-friendly experience. It is these laws of human nature that drove my decision to keep all user inputs on one side of a page, and all program outputs. The grouping of these by the user then simplifies the model into two sections, whereas if I had opted to separate everything all over the page, this would appear as a lot of options to process, which would be overwhelming for the user.

I have kept the damage radius of a potential meltdown on screen throughout the different stages of the program. This is so that the user continues to feel emersed in the situation they set up while making decisions on how to manage the meltdown. It will also allow them to see how their decisions impact outcomes of the situation. Following on from this, if my program is being used to train my stakeholders, it continues to provide scale and magnitude of the situation. When running through drills in any scenario it is easy to scale it down, as it isn’t real. It is important to train as if it is real and reminding the user of what’s at risk is an easy way to maintain this stress.

### Revised Program Designs

Chart

Description automatically generated

Figure - Revised Main Program Screen (Flowchart maker & online diagram software, no date)

Calendar

Description automatically generated

Figure - Revised Date Selection Window (Flowchart maker & online diagram software, no date)

Graphical user interface, application

Description automatically generated

Figure - Revised Wind Selection Window (Flowchart maker & online diagram software, no date)

#### Revised Main Program Screen

Having gotten well into programming my project I have had to make modifications to my original design. I have kept the structure of buttons to one side and map to the other side but swapped this around. I have also moved away from having date selection and as much information on the main screen and put most of the selection options in following pages. The primary reason for doing this was down to module limitations. For many cases it was easier to have another window open than try and cram all the programming into one file.

Along with this, it is also much tidier to separate windows into different files, as you can almost treat them as different programs and link them together at the end.

#### Revised Date Selection Window

Custom Tkinter Schimansky, T (no date) doesn’t have a built-in calendar so instead a separate module had to be used called Tkcalendar 1.5.0 documentation (no date). I was finding it difficult to implement this different module in the Custom Tkinter code of my main program, so instead decided to create a new window that ran off Tkinter Python interface to Tcl/Tk – Python 3.10.7 documentation (no date)

The user can select a date, and this is displayed in the main window.

#### Revised Wind Selection Window

I have decided to create an option to input the wind strength. This value is selected by the user and displayed on the main window. New window used for similar reasons as above, along with reducing clutter on main window.

### Algorithms

Figure - Main System Start Up (https://app.diagrams.net/)

#### Main System Start-up

My start-up module contains the steps that the program will follow to complete one full meltdown simulation. Following start-up of the program the user will be able to input the data for their desired simulation, these include offline or online mode, date, location, and severity. Once this data has been inputted the program will need to validate this data to ensure it can be used.

These checks include if the program is in online mode there must be weather data available for the day selected, or the location. If there isn’t then an error message will have to appear allowing the user to continue without the dynamic output or go back and choose a different date. This is given that the program is in online mode, if not then weather data doesn’t need to exist and the program will validate the data.

Flow diagrams outlining these steps are to follow.

I have made this main module to outline the overall function of my program so that I do not lose sight of the final product. It is important that I do not get lost down a rabbit hole when in programming and this module will help outline the overall process of my project.

#### Program Boot-up internet check

A yellow circle with black text

Description automatically generated with medium confidenceIn this module I outline the processes that take place when the program starts up. One of the main functions of my program looks at live weather data or past weather data of a place. For this to occur it is crucial that internet access is available, so upon booting up the program checks to see if the user is connected to the internet.

Figure - Internet Check (https://app.diagrams.net/)

A picture containing text, clipart

Description automatically generatedIf they are, then the program will continue as normal and give access to weather data for the user.

If they are not the program will go into offline mode and warn the user, then continuing to normal functionality. This option will also be available for the user to turn on and off should they wish to, even if internet access if available.

I have chosen to make this module as it saves the user time and effort, having to put in all the details and data needed to then be told the program cannot gain internet access is not a smooth user experience, this module removes this problem and allows the user to know immediately if there is an issue upon boot.

##### Pseudocode

Text

Description automatically generated

#### Online mode is on or off

This process outlines the processes the program will perform if internet access is available upon launch and the user goes to initiate a detonation. Following a connection to the internet, no changes will be made to the state of the online/offline switch, the map will update itself through the maps API, live data will be used to influence the output and the program will continue as normal.

I have made this module to ensure it is clear that no changes to the normal process will need to be made should the program be connected to the internet. This is important because if changes were made it would influence the users end result.

Figure - Online Mode is on (https://app.diagrams.net/)

A yellow circle with black text

Description automatically generated with medium confidenceA yellow circle with black text

Description automatically generated with medium confidence

This diagram outlines what will happen if upon bootup the user is not connected to the internet. Once the program has been unable to establish a connection a message will appear on screen informing the user they are not connected to the internet.

Once this has happened the program will automatically turn on offline mode, this means the program will skip any updates and live weather tracking it would normally do – as it cannot use the internet to check.

I have done it this way so that if the user connects to the internet, it is very easy to get back online on the program – just flick the switch. The message exists to tell the user why they won’t have access to some of the programs more advanced features, as they may not initially realised.

Figure - User not connected (https://app.diagrams.net/)

A picture containing text, clipart

Description automatically generated

A picture containing text, clipart

Description automatically generated

While these are similar to the previous flow chart, it allows me to section what exactly should happen in each eventuality. It is easy to say leave offline mode on/turn offline mode off, but what does this mean? How this happens is outlined above and in programming it allows to me to follow a clearer instruction of an expected process.

#### A picture containing text, clipart Description automatically generatedA picture containing text, clipart Description automatically generatedDiagram Description automatically generatedWeather Data Checker

##### Pseudocode

Figure - Detonate Sequence (https://app.diagrams.net/)

Text

Description automatically generatedThis process outlines how the program will firstly check whether weather data can be accessed. The user will begin by inputting a date for the meltdown to occur. I am currently unsure on how the weather data will be stored, and how much in the past I will be able to access. So, for now, the process will need to check whether data is present for the day chosen.

To start, the program will need to decide whether the day the user has chosen is in the future or the past, if in the future then the program won’t use weather data, as the conditions haven’t already happened. The user then will be able to decide if they want to choose another day that does have data available or continue this set day without a dynamic output.

If the day selected is in the past, then the program will try and access the data for this day, if available then it will continue and provide an output, if not then it will inform the user and give them the option to either continue this day without a dynamic output or choose a different day.

I have done this as many users will use my program with the hopes of dynamic outputs, and so if that isn’t possible for a parameter they can change with ease, I want it so that they can do this. However, if they don’t want to change the date, I also want to make it easy for them to continue without the dynamic output.

#### Detonation EventDiagram Description automatically generated

This process outlines what will happen upon the “Detonate” button being clicked on the main page. This procedure has many sub-procedures. This was done so that the code I had written was not too bulky and unorganised. To include this all-in-one function, it would make that section extremely long, and variable names may begin to overlap.

As can be seen, it will check to see if the location, date, and wind have all been chosen. If any of these variables have not been selected, then the program will run the missing data function and that process will begin.

Following all the data being good, it will run the radius calculation function. Here is where the dimensions of the polygon for the damage zone will be calculated.

Once this is completed, the polygon will be outputted, and the function will be completed.

##### Pseudocode

Text

Description automatically generated

Figure - Detonation Function (https://app.diagrams.net/)

#### Missing Data

Diagram

Description automatically generated

No

Figure - Missing Data (https://app.diagrams.net/)

Text

Description automatically generatedThis procedure outlines what will happen in the event that data the program looks for isn’t there. Some of the data the program could be given isn’t necessarily required for the final output, but some is, so different options are needed.

Should the data be required, a message informing the user will be produced, and they will be taken back to the main page. This is where this function ends.

If the data isn’t needed, it is possible that the user forgot to enter it, and so the program will warn them with the option to return and input it if they like or continue if not.

If they would like to input the data, it will go back and let them put it in. If not, the program will continue on and display the output.

#### Diagram Description automatically generatedCalculating Radius

This function are the steps the program takes when calculating the longitude and latitude points for the polygon.

The first step is to get the location of the marker that the user has placed. It will then split this apart at the comma and save the latitude and longitude into two separate variables.

It will then take these two variables and define them as floats, so I am able to manipulate them to the precision I need.

Once it has done this, then I am able to calculate each point depending on how far away it needs to be, plot the points on the map, and then output the polygon.

##### Pseudocode

Text

Description automatically generated

Figure - Radius Calculation (https://app.diagrams.net/)

#### Choosing Wind

Diagram

Description automatically generatedThis function outlines what will happen when the option to select how strong the wind will be is chosen. Upon the window being launched where the user selects the wind, the scale of the slider is set. This could be 1-10, or 1-100.

After this, the window is loaded, the user inputs their choice, the value is saved and the function ends.

I have done this to reduce the chances of error, allowing a user to input any data they liked would result in lots of incorrect data, whereas this way, the user can only input data which the program is expecting.

##### Pseudocode

Text

Description automatically generated

Figure - Choosing Wind (https://app.diagrams.net/)

#### Choosing Date

A picture containing text, sign, green

Description automatically generatedThis function outlines what happens when the user clicks the input date button.

It loads up a secondary window, this was done to reduce clutter on main page.

The user can then select the day, the value is saved, and the window is closed.

The function then finishes.

##### Pseudocode

Text

Description automatically generated

Figure - Date Choose (https://app.diagrams.net/)

### Key Variables

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable Name** | **Variable Type** | **Variable Use** | **Validation** |
| APP\_NAME | String | Stores the name of the main window | Cannot have incorrect data, name can be anything. Is declared in code. |
| WIDTH | Integer | Stores dimension for window width | Set in code |
| HEIGHT | Integer | Stores dimension for window height | Set in code |
| current\_position | String | Stores the coordinates for where the marker has been placed. | Is a string set by the tkintermapview module, needs to be converted into two separate float values for manipulation. |
| detonate\_position | String | Stores the coordinates for the centre of where the meltdown takes place. | A string defined by getting the value of the marker position. Cannot be incorrect data, is given by module. |
| latitude | Float | Stores the latitude of the marker position | Is previously split at the comma of both and stripped of the bracket (tkitnermapview sets the current\_position variable as (latitude,longitude) so for manipulation removal of characters is needed. It is also then defined as a float, as previously was defined as a string for manipulation). |
| longitude | Float | Stores the longitude of the marker position | Is previously split at the comma of both and stripped of the bracket (tkitnermapview sets the current\_position variable as (latitude,longitude) so for manipulation removal of characters is needed. It is also then defined as a float, as previously was defined as a string for manipulation). |
| pointxLa | Float | Is defining a certain point of the polygon drawing the damage radius. x changes for each point, is a 1 for the first point, 2 for the second and so on. It is defined as the latitude of the marker position plus or minus whatever value is needed to reach the distance away that there would be damage. | Is only defined by values given by module, so incorrect data worries are not applicable. Conversion between miles to latitude scale is important here to ensure the point is plotted the correct distance away. |
| pointxLo | Float | Is defining a certain point of the polygon drawing the damage radius. x changes for each point, is a 1 for the first point, 2 for the second and so on. It is defined as the longitude of the marker position plus or minus whatever value is needed to reach the distance away that there would be damage. | Is only defined by values given by module, so incorrect data worries are not applicable. Conversion between miles to longitude scale is important here to ensure the point is plotted the correct distance away. |
| windchoice | Integer | This stores the value of the slider when the user is selecting the wind strength. | To remove the chances of incorrect data the user is only able to use a slider with pre-set values to select the wind strength. This means they can only provide data the program is expecting. |
| datechoice | String | This stores the value of the calendar when the user is selecting the date. | Similar to the slider, this has been done to remove any issues with incorrect data. It also removes any worry of differences in international date formatting, as it is all selected through a calendar window. |
| longstraight | Integer | To store the longitude value which each reactor will use to plot the sidewise points of the polygon. | Cannot have incorrect values, as set in program and not edited. |
| latstraight | Integer | To store the latitude value which each reactor will use to plot the vertical points of the polygon. | Cannot have incorrect values, as set in program and not edited. |
| longsmall | Integer | This stores the longitude value used to plot the corner points of the polygon. | Cannot have incorrect values, as set in program, and not edited. |
| latsmall | Integer | This stores the latitude value used to plot the corner points of the polygon. | Cannot have incorrect values, as set in program, and not edited. |
| choice | String | Stores the user’s choice of which type of reactor. | Is valued via a dropdown menu with preset options, cannot have incorrect data. |

## Test Data

Throughout the development process, while undergoing iterative development, I will mostly use random and extreme data. This will be to test both the logic and limits of my program. For example, when selecting the wind, I will look at all wind strengths, but when simply making sure the logic works, I will use the largest wind strength each time. With date selection, I will either leave it blank or select a day at random until I get to the point where the date is influencing the output, at which point I will find a day where the wind was bad and use the to ensure I can see results.

However, after the programming is complete, I will go through and very methodically test each section of my program with expected and erroneous data, to ensure that the program is handling and outputting everything correctly. I will also compare its outputs to online sources to make sure that the size of damage is correct.

The reason I have chosen to test my program in this way is so that I can focus on getting the syntax and logic of my program done first, to build up the foundation and walls of the program and ensure that the fundamentals are working. Following the successful completion of this, I will then move on to refining the details of my code and making sure that it works to the exact expected specification.

## Test Table Plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No.** | **What is Being Tested** | **Type of Test** | **Input Data or Actions** | **Expected Outcome** |
|  | An independent window loads up, with all aspects correctly shown and positioned. | Typical | Program is ran. | World map on the right, fully interactive with up-to-date placenames, and all buttons and labels appearing on the left correctly. |
|  | The marker can be placed, and its coordinates are printed. | Typical | Place Marker button is pressed. | A marker is placed in the centre of the map and the coordinates are printed. |
|  | The marker is updated in a new position. | Typical | Map is moved and Place Marker button is pressed. | Upon moving the map (the centre of the screen changes location) and pressing Place Marker, the original marker is deleted and a new one is set. |
|  | Choose Date button outputs a calendar window. | Typical | Choose Date button is pressed. | Upon clicking the “Choose Date” button, a new window pops up allowing the user to select the date they wish from a calendar. |
|  | If a date within 5 days of the current date is selected, weather predictions from the internet are used for weather. | Typical | Date within 5 days of the future is selected. | If the user selects a day that is 5 days within the future, the program will gather weather data from the internet and use that to input the wind strength and tell the user that the data used are predictions only. |
|  | If a date beyond 5 days in the future is selected, program doesn’t fail. | Erroneous | Beyond 5 days into the future is selected. | When the user selects a day further than 5 days in the future is selected, the program will give the user the option to either select custom data or select another day. |
|  | The date is outputted onto the main screen. | Typical | User selects date. | Once the user chooses a day and clicks “Get Date” this date is outputted onto the main window and is overwritten if a different day is chosen. |
|  | Wind Strength button outputs a wind strength slider. | Typical | Wind Strength button is pressed. | Once the user clicks the “Wind Strength” button, a window pops up allowing the user to manipulate a slider and choose the wind strength. |
|  | The wind strength is outputted onto the main screen. | Typical | User selects wind strength. | When the user selects the wind strength this is displayed onto the main screen and overwritten if the user changes their mind. |
|  | Maximum/minimum wind strength outputs maximum/minimum radius. | Boundary | User uses max/min wind strengths. | When the user inputs the maximum or minimum wind strengths you can using the slider the program outputs either the largest effect on the output or no effect on the output (for minimum) |
|  | The dropdown menu displays different options | Typical | User clicks dropdown menu | Upon clicking the dropdown menu, it will display different options the user can choose. |
|  | Upon clicking detonate a damage radius is outputted and the next window is displayed. | Typical | User clicks “Detonate” button. | When the user clicks the detonate button, the damage radius of a given reactor melting down is displayed, with the middle being where the maker was placed, and a second window appears. |
|  | The secondary window shuts down correctly. | Typical | User follows buttons through to closing second window or uses X to close it. | Once the secondary window is shut, the main program should not change. The radius of damage should remain, and the user should be able to continue to change variables and output more damage. |
|  | Secondary window closes when main window is closed. | Erroneous | Main window is closed before secondary. | Both windows close correctly. |
|  | Upon clicking detonate again, another window opens, and the original damage radius isn’t removed. | Erroneous | User clicks Detonate again, not on a fresh window. | If the user clicks detonate again, another damage circle appears and if the secondary window is already running, it will not close. |
|  | A circle of damage is outputted when “Detonate” is pressed | Typical | Detonate button is pressed. | Once the location has been set, should no other variables impact it, the damage radius should be a perfect circle, as a average model will have an exclusion zone the same amount of distance from the centre point/nuclear meltdown. |
|  | Choosing the date automatically submits the wind strength for that day if available. | Typical | Date is selected. | Once the user selects the date the program will look at the weather data for that day and automatically input the average wind strength on that day for that location. |
|  | Wind strength will increase the damage radius in whatever direction it was blowing. | Typical | Wind strength is submitted when Detonation occurs. | When the wind strength is submitted, either by the user or the date selection, the radius of damage is increased in whatever direction said wind is blowing, as wind will blow the radiation from a meltdown further. |
|  | The different types of reactors in the drop-down menu create differently sized damage radii. | Boundary | Different options are selected by the user and detonate button is pressed. | Should one type of meltdown be worse than another, the radius of damage will be larger, and the inverse for a meltdown that is less severe. |
|  | There are interactive sliders in the secondary window. | Typical | User clicks Detonate button. | Upon the second window launching there are four interactive sliders the user can manipulate. |
|  | The second window is an interactive game that lets you control the reactor. | Typical | User slider manipulation. | Once this second window launches the user will be able to use the sliders to control different aspects of a nuclear reactor and attempt to bring it back from a meltdown. |
|  | The red/amber/green display light | Typical and Boundary | User interacts with the sliders. | When the sliders are used, if the effect had been good, the light will go green, no change is amber, and if the affect is bad then red. Typically setting any value to its maximum will have a bad value and so this will be the boundary test too.  Maximum values will (given that it is true) result in a red light. |
|  | The program runs without a stream of errors in my Visual Studio output window. | Typical | The program is ran. | Upon running the program, my code output window will only output something when it is intended to, no errors are shown. |
|  | Upon loading up my program pings the internet to ensure user is connected. | Typical | Program is loaded up | Upon the program loading up it will test to see if the user is connected to the internet, and display whether they are or not in a small text window in the screen. This will eventually serve the purpose of deciding whether the user is able to access the internet or not the use realtime weather data in the program. |
|  | Offline mode switch functions correctly | Typical | Switch is clicked | Upon the switch being clicked the program will go into offline mode, this means realtime weather data will not be used. This is a decision the user can make if they do not want this to happen. |
|  | Program warns of no internet correctly | Erroneous | Program runs detonate sequence without internet | Should the user try and run the detonate sequence without internet access instead of breaking, the program will simply display a warning message asking the user to try again. |
|  | Control of all four sliders on second window is accurate and dynamically affects gameplay. | Typical | User interacts with sliders on second window | Usage of the sliders accurately updates values which are in a check loop, and once reaching a certain threshold display either success or failure in terms of whether the user was able to bring the controller back under control or not. |
|  | Sliders are set to maximum/minimum | Boundary | User sliders any or all sliders to maximum or minimum values | This should have no outstanding effect on the program, only having the most or least effect on the status of the reactor core, and will result in a quicker or slower regain of control, however, typically maxing out or removing any of the parameters in this situation will have overwhelmingly negative effects on the cores status. |
|  | Success screen appears with correct buttons | Typical | User successfully brings reactor back under control | Once the user has been able to successfully regain control over the reactors core, a success screen appears. This will have the option to return to the map page or quit the application. |
|  | Failure screen appears with correct buttons | Typical | User unsuccessfully brings reactor back under control | If the user is unable to bring the reactor back under control, then the failure screen appears. This will have the option to return to the map screen, try again, or quit the application. |
|  | Reset button removes all polygons | Typical | User selects reset button. | When the user selects the reset button, all polygons on the screen are removed. |
|  | If no reactor type is selected, a warning message appears. | Erroneous | User tries to detonate without selecting reactor type | Should the user run the detonate button without selecting a reactor type, then instead of the variables missing values and an error occurring, the program instead warns the user, and asks them to input a type. |

## Summary

Following the completion of my Design the scope of my project has taken shape. I am happy that I have developed a solid framework for the final product I intend to produce. As I have began working on the project key aspects of my Design have changed, and so revised sections have been included to reflect this. These have been clearly labelled and summarized.

As I continue work on this project the approach to the Software Development Life Cycle will change. Up until now, I have been following a traditional Waterfall approach, whereby I have focused on one section, developed its look and planned out how it will work. Following this, I have communicated with my stakeholders to receive feedback and any adjustments that need to be made. Once this has been done, I will continue working on this section. Finally, once I have done this for one section, I will move onto the next.

As I move onto the development of the program, I will change my approach to the Software Development Life Cycle to an Iterative Approach. This is where I will go through the steps of developing different sections of the program, receiving feedback and then repeating this with my new requirements.

I will remain in good contact with my Stakeholders, consistently and repeatedly getting feedback and changing aspects of my program that need adjustments. Should things need changing, hopefully this will happen before they are fully finished, reducing wasted time and effort.

At points during the early development section, it became apparent that it would be easier to format the design of my program differently to what I had originally planned. This was down to ease of coding, and not wanting to change thing around if not needed be, I communicated with my Stakeholders to ensure this worked with them and revised the Wireframe Diagrams produced in my Design Phase.

Following all this, I feel I am now ready to move onto the programming section. My Stakeholders are also happy, and I will continue to keep in touch with them throughout this process.

# Developing the Coded Solution

Max (25 Marks)

## Prototype 1

Prototype 1 of my program will not be the completed product of my program. Rather than focus on the final parts of my program, a large focus has been on ensuring that the UI is working correctly, in the manner I want it to, finding suitable modules that allow for the adaptability I require, along with getting together the fundamental maths required to accurately output a damage radius that can be easily understood by the user.

Not as much progress has been made on Prototype 1 as originally hoped and expected. I am at a stage where the program can successfully output the damage radius of a Chernobyl type nuclear meltdown to a level of plausible accuracy anywhere in the world, however, there is a lot still missing from my program.

This is due to nearly every single aspect of my program so far having been a completely new concept I have needed to learn. It has taken a while to fine tune everything, ensure everything is correctly communicating and working together, and is fundamentally correct. Having taken everything on board it should now be much easier to add more details to my program, now that the framework is complete.

The next Prototypes will focus on introducing different types of meltdowns, wind impacting the radius of damage, and automatic weather grabbing for different days. Each of these will be introduced in turn and will have adjustments made as the Stakeholders provide input and adjustments during my Iterative Approach to the Software Development Life Cycle.

### Text Description automatically generatedWritten Code

Figure - Setting a marker

This code is using the tkintermapview module and upon the placement of a marker on the map, will get the position of this marker and save it under current\_position as a latitude and longitude coordinate. It will then add this to an array of positions saved which would allow me to introduce multiple markers on the map at once, however for now, I will not implement this feature to reduce clutter on the map.

I have chosen to do this for now to reduce clutter on the map, the polygon of damage remains, however there will only be one marker on the map at a time, wherever the user places it. This also means they are much better able to read the names of places in the radius, that are no longer being blocked by the marker.

It will then print the position on the map to allow me to ensure the data I am expecting is correct, and to see what format it comes out in.

Text

Description automatically generated

This code will take the longitude and latitude (in the format (longitude, latitude)) from the code above, and split it at the comma into two separate variables.

It will the strip each of those variables of their respective open and closed bracket and define the remaining number as a float which I can then manipulate in future code. I then print this out so that I am able to follow what my code is doing and to ensure it has correctly split and stripped each variable.

Figure - Defining latitude and longitude

Little decisions have had to be made here, everything I have done is necessary to manipulate and use the data given.

Chart

Description automatically generated This code plots each point of the polygon which represents the damage radius of a nuclear meltdown. Each point is defined by taking the longitude and latitude of the marker position, and adding or subtracting a certain value from them, which equates to the distance in miles that would be affected. The more points I plot the more accurate the damage radius, however each point is the exact distance from the centre marker.

Figure - Setting polygon points

Text

Description automatically generatedCurrently I build an Octagon, as for each point I must perform a new calculation and with the interest of time in mind, and Octagon represented the closest shape to a circle without needing an extremely large number of calculations to complete.

Figure - Second page sliders

Upon clicking detonate, you are met with a secondary window containing some text and four sliders. Eventually this will be where you can control certain aspects of the reactor but for now this code is redundant, a preview of what is to come.

I have chosen to use sliders as they provide a good balance between the user having a fair amount of control over the system but aren’t overwhelmed by options and buttons. It also means I will not have to process incorrect data, as the users can only input data through the sliders themselves.

I will also have a responsive display next to each slider representing whether the level it has reached is good or bad, allowing the user to adjust their decisions based on the feedback they are getting. This is all planned to come in later prototypes.

Text

Description automatically generated

Figure - Defining wind choice

This is where the windchoice variable is set. Following the user selecting their wind strength with a slider, the value this returns is saved and set as a global variable to be used later on.

Text

Description automatically generated

Figure - Defining date choice

Similar to above, this does the same but for the date the user selects instead.



Figure - Setting up main window

This code sets up the window for the main code with its title and dimensions. The dimensions are predefined above with no real reason for the values. It was a good fit and well displayed everything I needed to show, and if this it not big/small enough for the user, the window can easily be resized.



Figure - Frame layout

This code sets up the layout of my main window. The top section defines the left-hand frame, where all my buttons are positioned.

The left-hand frame is in the first column and the right-hand frame is in the second column, the left-hand frame is significantly thinner than the right hand frame. This is so that the map has the majority of the room on the page, as this is where a majority of the user’s attention will be based, and is also the most detail intensive part of the application.



Figure - Positioning the map

This is where the Map is inserted into the window. Placed in the right-hand frame, it is the main bulk of the main page.

### Testing

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No.** | **What is Being Tested** | **Type of Test** | **Input Data or Actions** | **Expected Outcome** | **Actual Outcome** |
| 1. | An independent window loads up, with all aspects correctly shown and positioned. | Typical | Program is ran. | World map on the right, fully interactive with up-to-date placenames, and all buttons and labels appearing on the left correctly. | PASS |
|  | The marker can be placed, and its coordinates are printed. | Typical | Place Marker button is pressed. | A marker is placed in the centre of the map and the coordinates are printed. | PASS |
|  | The marker is updated in a new position. | Typical | Map is moved and Place Marker button is pressed. | Upon moving the map (the centre of the screen changes location) and pressing Place Marker, the original marker is deleted and a new one is set. | PASS |
|  | Choose Date button outputs a calendar window. | Typical | Choose Date button is pressed. | Upon clicking the “Choose Date” button, a new window pops up allowing the user to select the date they wish from a calendar. | PASS |
|  | If a date within 5 days of the current date is selected, weather predictions from the internet are used for weather. | Typical | Date within 5 days of the future is selected. | If the user selects a day that is 5 days within the future, the program will gather weather data from the internet and use that to input the wind strength and tell the user that the data used are predictions only. | FAIL  Currently the date selection has no affect on the program. When the user selects the day, currently it does not grab the weather and affect the output of the program.  It will eventually scrape weather predictinons off the internet and use those predict where the damage will be.    (Current Date: 17/10/2022) |
|  | If a date beyond 5 days in the future is selected, program doesn’t fail. | Erroneous | Beyond 5 days into the future is selected. | When the user selects a day further than 5 days in the future is selected, the program will give the user the option to either select custom data or select another day. | FAIL  Currently – same as in test 5 – the selected date has no affect on the output of the program. If the user selects a day in the future without any preidcitons, the program will not do as expected, therefore failing this test.    (Current Date: 17/10/2022) |
|  | The date is outputted onto the main screen. | Typical | User selects date. | Once the user chooses a day and clicks “Get Date” this date is outputted onto the main window and is overwritten if a different day is chosen. | PASS |
|  | Wind Strength button outputs a wind strength slider. | Typical | Wind Strength button is pressed. | Once the user clicks the “Wind Strength” button, a window pops up allowing the user to manipulate a slider and choose the wind strength. | PASS |
|  | The wind strength is outputted onto the main screen. | Typical | User selects wind strength. | When the user selects the wind strength this is displayed onto the main screen and overwritten if the user changes their mind. | PASS |
|  | Maximum/minimum wind strength outputs maximum/minimum radius. | Boundary | User uses max/min wind strengths. | When the user inputs the maximum or minimum wind strengths you can using the slider the program outputs either the largest effect on the output or no effect on the output (for minimum) | FAIL  Currently there is no effect on the output based on what wind strength the user inputs, and so there is no correct behavour from the program. |
|  | The dropdown menu displays different options | Typical | User clicks dropdown menu | Upon clicking the dropdown menu, it will display different options the user can choose. | PASS |
|  | Upon clicking detonate a damage radius is outputted and the next window is displayed. | Typical | User clicks “Detonate” button. | When the user clicks the detonate button, the damage radius of a given reactor melting down is displayed, with the middle being where the maker was placed, and a second window appears. | PASS |
|  | The secondary window shuts down correctly. | Typical | User follows buttons through to closing second window or uses X to close it. | Once the secondary window is shut, the main program should not change. The radius of damage should remain, and the user should be able to continue to change variables and output more damage. | PASS |
|  | Secondary window closes when main window is closed. | Erroneous | Main window is closed before secondary. | Both windows close correctly. | PASS Unable to show screenshot. |
|  | Upon clicking detonate again, another window opens, and the original damage radius isn’t removed. | Erroneous | User clicks Detonate again, not on a fresh window. | If the user clicks detonate again, another damage circle appears and if the secondary window is already running, it will not close. | PASS |
|  | A circle of damage is outputted when “Detonate is pressed” | Typical | Detonate button is pressed. | Once the location has been set, should no other variables impact it, the damage radius should be a perfect circle, as a average model will have an exclusion zone the same amount of distance from the centre point/nuclear meltdown. | FAIL    Currently the program produces an octagon. The module I am using allows you to place polygons on the map, not shapes. This means each point needs to be plotted. While there is a way to automate the plotting of hundreds of points, each point needs a longitude and latitutde coordinate, taken from the location of the marker, which both have different scales and need converting to from miles, fundamentally a circle is not possible, but on top of that, the next best thing will need a serious quantity of mathematical forethought, something that I lack the skills and time for to introduce in Prototype 1. |
|  | Choosing the date automatically submits the wind strength for that day if available. | Typical | Date is selected. | Once the user selects the date the program will look at the weather data for that day and automatically input the average wind strength on that day for that location. | FAIL    Due to time limitations, and other, more fundamental, aspects of the project taking priority, this feature is yet to be introduced.  Some form of weather data API will be required to grab the weather data for a specified day, or a large spreadsheet containing the wind strength for as much of history as possible across the glone will be neeeded to achieve this, and is quite a large scale feature to implement. |
|  | Wind strength will increase the damage radius in whatever direction it was blowing. | Typical | Wind strength is submitted when Detonation occurs. | When the wind strength is submitted, either by the user or the date selection, the radius of damage is increased in whatever direction said wind is blowing, as wind will blow the radiation from a meltdown further. | FAIL    Research into how much wind will blow the radiation from a meltdown needs to be done and stored in a database.  Following this, each type of meltdown needs to have a specific multiplier for its own plotted points based upon how much this wind strenght will increase where they sit.  Again, a time consuming task that I did not have time for in this Prototype. |
|  | The different types of reactors in the drop-down menu create differently sized damage radii. | Boundary | Different options are selected by the user and detonate button is pressed. | Should one type of meltdown be worse than another, the radius of damage will be larger, and the inverse for a meltdown that is less severe. | FAIL      The maths to plot the points for one polygon is currently all manual (individual plus and minus of preset values) and needs to be adapted so the number being added on or taken away is a variable.  This variable then needs to be set depeding on which type of reactor has been chosen.  To do this, each type of reactor needs an average exclusion zone (area of damage) which then needs to be converted into four longitude and latitude values, one of each that goes straigh up, straight down, straight left, and stragiht right and then another two that draws the two shorter sides of the triangle to reach the other points on the octagon.  This will not take long, but there simply wasn’t time to include in this prototype. |
|  | There are interactive sliders in the secondary window. | Typical | User clicks Detonate button. | Upon the second window launching there are four interactive sliders the user can manipulate. | PASS |
|  | The second window is an interactive game that lets you control the reactor. | Typical | User slider manipulation. | Once this second window launches the user will be able to use the sliders to control different aspects of a nuclear reactor and attempt to bring it back from a meltdown. | FAIL  Currently this entire aspect of the project has not been started.  The focus has been, and for the time being will continue to be, on the main window and correctly outputting dynamic damage zones.  Once this is finished my focus will shift to this second section, however, I felt no need to list tests for a section yet to exist, and so further development will come in later Prototypes. |
|  | The program runs without a stream of errors in my Visual Studio output window. | Typical | The program is ran. | Upon running the program, my code output window will only output something when it is intended to, no errors are shown. | FAIL |

### Plans for Prototype 2

As mentioned, Prototype 2 will see the addition of a more developed choice menu for the type of meltdowns. As can be seen in test 15 currently choosing different types of reactors has no effect on the output of the program. I would like to have at least 4 different occurrences, each with a different radius of damage, varying from larger to smaller than my current only option. Along with this, I may look at trying to automate a system which increases the number of points I have on my polygon to get it closer to a circle. This would be a complicated process due to the difference in scales between longitude and latitude, but something I believe I can achieve in future prototypes.

Test 14 outlines how currently adjusting the wind level does not have any effect on the output of the damage. In my following prototypes I want to be able to adjust the wind and this influence the radius of damage, however, this currently isn’t working. I am not sure whether it will be possible to achieve this for Prototype 2 but I do intend to have this complete for my final solution.

Test 13 similarly outlines how I want the choice of date to grab the wind of that day in the real world and have this affect my output. Again, this currently isn’t a feature I have been able to design, and I won’t be focusing on getting this completed for Prototype 2 as there are more fundamental aspects I intend to complete, but it is a part of the program I will be working on in the background.

I will also look at sitting different polygons on top of each other which are different colours, these will represent different severities of damage that would be done by a meltdown and would all appear at the same time upon detonate being clicked. I will also include a reset button on the main window, this will wipe the map clean of any existing polygons and allow the user to continue doing things without having to close the window and open it again.

As can be seen in Test 18, at points my code outputs a stream of errors coming from one of the APIs I am using. I am yet to identify which one, but I believe it is to do with my map view API. This could be an error I cannot fix as it is not my code, however, in Prototype 2 I will look at where these are coming from an attempt to reduce/fix them.

## Prototype 2

Prototype 2 will focus on the aspects of Prototype 1 I was unable to fully finish and polish off, rather than focusing on including major new aspects of the program. The main additions have been outlined in my plan for this prototype in prototype 1, so I will not go into detail here. Timings have been much more limited for the completion of this section, hence why this decision has been made, due to being away for a big chunk of the time where progress could’ve been made.

This was something I was aware of ahead of time, so the plan for this prototype could be accurately adjusted and my stakeholders have been informed accordingly. This has allowed me to nail down the features I wanted to complete and get these done, and larger aspects of the program will be completed in future prototypes.

### Algorithms

#### Reactor Selection

Diagram

Description automatically generated This function is designed to take the user input decision for the type of reactor that will meltdown, ensure a selection has actually been made, and then send this decision to the next function.

Figure - Setting reactor type

The next function is reactortypedata() which will actually be a collection of functions, decided by an if statement depending on the user’s decision, which will then set the variables used to plot the points of the damage radius polygon on the map.

##### Pseudocode

Text

Description automatically generated

#### Reactor Type Data Set

A picture containing text, sign

Description automatically generatedThis function will be what the function outlined in Figure 32 will go to. There will be a few of these, each following the same steps, depending on how many different types of reactors I have.

The only part that will differ is the value that the variables (in green boxes) are set to, which will depend on how big or small of a meltdown there is.

Once these values are set, the program will the run the main part of the code that plots the points, which will use the values this function has set out.

##### Pseudocode

Text

Description automatically generated

Figure - Setting data for reactor type (app.diagrams.net)

#### Reset button

Diagram

Description automatically generatedThis program outlines the reset button. Simply, upon the user clicking the reset button on the main window, and existing polygons will be removed from the map.

Figure - Reset button (app.diagrams.net)

##### Pseudocode

Text

Description automatically generated

#### Program Internet Ping

Diagram

Description automatically generated

Figure - Internet ping (app.diagrams.net)

This function outlines the process of pinging the internet and what the program will do with a successful/unsuccessful ping. If connected, a small “Connected” message will appear in the bottom of the main page indicating a successful connection.

If the program runs and is not connected to the internet, then a warning message will appear to the user, letting them know, and then a small “Disconnected” message will appear in the same place at the bottom of the main page.

No internet connection will also enable offline mode, which will remove any form of real time weather data from the program.

##### Pseudocode

Text

Description automatically generated

### Written Code

Text

Description automatically generated

Figure - Clearing marker and polygons

The above three functions are what allows me to remove the polygons and markers once I click the reset button built in a future section of code. The set\_marker\_event() code is what allows me to set the red marker and runs first the clear\_marker() function. All this does is remove the first marker and returns to the original function and places a new one. I am still undecided whether to keep the red marker in the middle of my polygons to mark the centre, or whether to remove them. For now, I am removing them to reduce clutter, but I have considered keeping them in to mark where the reactor would be.

Should the reset button be clicked then the clear\_marker\_event() will run, this first goes through and deletes and red markers in the list (list made when marker is placed down) and then runs through the list of polygons that have been placed (polygons are added to an array upon creation) and deletes them. Using an array to store made polygons and markers was the simplest way of achieving my goal and makes manipulating them very easy.

Text

Description automatically generatedThis code is what sets the various values for each of the options of meltdown. In the main program you are able to select which type of reactor you wish to meltdown and depending on your choice the values for the polygon points are set.

Figure - Setting data

The longstraight and latstraight variables are what set the top and bottom, and left and right points, and the two small variables are what set the corner points.

This code is easily extendable, with simple maths needed to calculate the new points.

Text

Description automatically generatedThis is the code that the code in Figure 37 defines the variables for. Previously, the longstraight, latstraight, longsmall, and latsmall variables were all integer values, however, now I have more than one reactor option, these are set as values prior to using them. This allows for reduced repeats of my code, along with ease of adding new reactor types. All I need to do to achieve this is add another if statement (see Figure 37) and calculate the new values.

Figure - Variable Replacement

### Testing

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No.** | **What is Being Tested** | **Type of Test** | **Input Data or Actions** | **Expected Outcome** | **Actual Outcome** |
|  | An independent window loads up, with all aspects correctly shown and positioned. | Typical | Program is ran. | World map on the right, fully interactive with up-to-date placenames, and all buttons and labels appearing on the left correctly. | PASS |
|  | The marker can be placed, and its coordinates are printed. | Typical | Place Marker button is pressed. | A marker is placed in the centre of the map and the coordinates are printed. | PASS |
|  | The marker is updated in a new position. | Typical | Map is moved and Place Marker button is pressed. | Upon moving the map (the centre of the screen changes location) and pressing Place Marker, the original marker is deleted and a new one is set. | PASS |
|  | Choose Date button outputs a calendar window. | Typical | Choose Date button is pressed. | Upon clicking the “Choose Date” button, a new window pops up allowing the user to select the date they wish from a calendar. | PASS |
|  | If a date within 5 days of the current date is selected, weather predictions from the internet are used for weather. | Typical | Date within 5 days of the future is selected. | If the user selects a day that is 5 days within the future, the program will gather weather data from the internet and use that to input the wind strength and tell the user that the data used are predictions only. | FAIL  Currently the date selection has no affect on the program. When the user selects the day, currently it does not grab the weather and affect the output of the program.  It will eventually scrape weather predictinons off the internet and use those predict where the damage will be.    (Date of test: 17/10/2022) |
|  | The wind strength is outputted onto the main screen. | Typical | User selects wind strength. | When the user selects the wind strength this is displayed onto the main screen and overwritten if the user changes their mind. | PASS  Graphical user interface, text, application, chat or text message  Description automatically generated  Graphical user interface, text, application, chat or text message  Description automatically generated |
|  | Maximum/minimum wind strength outputs maximum/minimum radius. | Boundary | User uses max/min wind strengths. | When the user inputs the maximum or minimum wind strengths you can using the slider the program outputs either the largest effect on the output or no effect on the output (for minimum) | FAIL  Currently there is no effect on the output based on what wind strength the user inputs, and so there is no correct behavour from the program. |
|  | The dropdown menu displays different options | Typical | User clicks dropdown menu | Upon clicking the dropdown menu, it will display different options the user can choose. | PASS |
|  | Upon clicking detonate a damage radius is outputted and the next window is displayed. | Typical | User clicks “Detonate” button. | When the user clicks the detonate button, the damage radius of a given reactor melting down is displayed, with the middle being where the maker was placed, and a second window appears. | FAIL  In Test 18 to remove the error stream I have stopped this function from running. At the moment it serves no purpose, and so there was no need to have it run if it was only causing issues.  When I overhaul the function I hope to get this feature working agauin. |
|  | The secondary window shuts down correctly. | Typical | User follows buttons through to closing second window or uses X to close it. | Once the secondary window is shut, the main program should not change. The radius of damage should remain, and the user should be able to continue to change variables and output more damage. | FAIL  The secondary window currently does not launch, and so cannot close down. |
|  | Secondary window closes when main window is closed. | Erroneous | Main window is closed before secondary. | Both windows close correctly. | PARTIAL PASS  Second window doesn’t launch, so cannot close, but main window does close properly. |
|  | Upon clicking detonate again, another window opens, and the original damage radius isn’t removed. | Erroneous | User clicks Detonate again, not on a fresh window. | If the user clicks detonate again, another damage circle appears and if the secondary window is already running, it will not close. | PARTIAL PASS    The second radius does appear correctly, however, the second window is curretly a postponed feature, and doesn’t launch to ammend the error fixed in Test 18. |
|  | A circle of damage is outputted when “Detonate is pressed” | Typical | Detonate button is pressed. | Once the location has been set, should no other variables impact it, the damage radius should be a perfect circle, as a average model will have an exclusion zone the same amount of distance from the centre point/nuclear meltdown. | FAIL  Chart, map  Description automatically generated  Currently the program produces an octagon. The module I am using allows you to place polygons on the map, not shapes. This means each point needs to be plotted. While there is a way to automate the plotting of hundreds of points, each point needs a longitude and latitutde coordinate, taken from the location of the marker, which both have different scales and need converting to from miles, fundamentally a circle is not possible, but on top of that, the next best thing will need a serious quantity of mathematical forethought, something that I lack the skills and time for to introduce in Prototype 1. |
|  | Choosing the date automatically submits the wind strength for that day if available. | Typical | Date is selected. | Once the user selects the date the program will look at the weather data for that day and automatically input the average wind strength on that day for that location. | FAIL  A screenshot of a phone  Description automatically generated with medium confidence  Due to time limitations, and other, more fundamental, aspects of the project taking priority, this feature is yet to be introduced.  Some form of weather data API will be required to grab the weather data for a specified day, or a large spreadsheet containing the wind strength for as much of history as possible across the glone will be neeeded to achieve this, and is quite a large scale feature to implement. |
|  | Wind strength will increase the damage radius in whatever direction it was blowing. | Typical | Wind strength is submitted when Detonation occurs. | When the wind strength is submitted, either by the user or the date selection, the radius of damage is increased in whatever direction said wind is blowing, as wind will blow the radiation from a meltdown further. | FAIL  Map  Description automatically generated  Research into how much wind will blow the radiation from a meltdown needs to be done and stored in a database.  Following this, each type of meltdown needs to have a specific multiplier for its own plotted points based upon how much this wind strenght will increase where they sit.  Again, a time consuming task that I did not have time for in this Prototype. |
|  | The different types of reactors in the drop-down menu create differently sized damage radii. | Boundary | Different options are selected by the user and detonate button is pressed. | Should one type of meltdown be worse than another, the radius of damage will be larger, and the inverse for a meltdown that is less severe. | PASS |
|  | There are interactive sliders in the secondary window. | Typical | User clicks Detonate button. | Upon the second window launching there are four interactive sliders the user can manipulate. | FAIL  Currently the second window does not load in, a feature removed to achieve a Partial Pass in Test 18. |
|  | The second window is an interactive game that lets you control the reactor. | Typical | User slider manipulation. | Once this second window launches the user will be able to use the sliders to control different aspects of a nuclear reactor and attempt to bring it back from a meltdown. | FAIL  Currently this entire aspect of the project has not been started.  The focus has been, and for the time being will continue to be, on the main window and correctly outputting dynamic damage zones.  Once this is finished my focus will shift to this second section, however, I felt no need to list tests for a section yet to exist, and so further development will come in later Prototypes. |
|  | The program runs without a stream of errors in my Visual Studio output window. | Typical | The program is ran. | Upon running the program, my code output window will only output something when it is intended to, no errors are shown. | PARTIAL PASS    Currently the error stream that once ran in my output box upon clicking the detonate button is no longer happening. This is good, however, not there completely.  I have achieved this by temportatily removing the launch of the second window when the detonate button is clicked, which indeed has removed the erorrs, but this is a feature of my program.  This tells me the error is contained in the second window, and has something to do with it launching and closing. The fix may be to use Custom Tkinter with this window, rather than tkinter, however, this will be looked at further in the next prototype. |
|  | Reset button removes all polygons | Typical | User selects reset button. | When the user selects the reset button, all polygons on the screen are removed. | PASS |
|  | Upon loading up my program pings the internet to ensure user is connected. | Typical | Program is loaded up | Upon the program loading up it will test to see if the user is connected to the internet, and display whether they are or not in a small text window in the screen. This will eventually serve the purpose of deciding whether the user is able to access the internet or not the use real time weather data in the program. | FAIL  This feature was not able to be completed for this Prototype |
|  | If no reactor type is selected, it will default to smallest one. | Erroneous | User tries to detonate without selecting reactor type | Should the user run the detonate button without selecting a reactor type, then instead of the variables missing values and an error occurring, the program instead sets the reactor to the default lowest value. | PASS |

### Plans for Prototype 3

In this Prototype, I want to get my second window running without an error stream at any point. As seen in Test 18 I have temporarily fixed this by stopping the launch of the second window, however, I want no errors with the launch of the second window. I believe it has something to do with the module I have used, and so will be overhauling the code into Custom Tkinter instead. This is what I have used for the rest of the program, and I believe it will work better. This is not the main aspect of this Prototype but is one thing I will work on for it.

Test 3 looks at how currently when a new marker is placed, the original one is deleted. For Prototype 3 I would like to change this so that the original marker is kept. These are acting as the spot where the Nuclear Powerplant would sit, and so if the radius is kept, the marker should stay too. Upon clicking the “Reset” button however, all of these markers would be removed too.

Unfortunately, I was unable to complete the pinging internet feature I had hoped to include in my Prototype 2, and so instead will be pushing this back to Prototype 3. The reset button and multiple options features took longer than previously anticipated, and so I was unable to get this done. I would also like to include the switch in this prototype. It is unlikely that it will be functional, as that would depend on interactive weather being implemented. However, I would like to get the switch itself sorted, and have it change the Connected/Disconnected button on the main page.

## Prototype 3

### Algorithms

### Written Code

### Testing

### Plans for Prototype 4

# Table of Figures

[Figure 1 - Nuclear Reactor Simulator (no date) Nuclearinst.com 6](file:///C:\Users\joshu\Documents\GitHub\Programming-Project\Joshua%20Netherton%20-%20Component%203.docx#_Toc118722665)

[Figure 2 - Control Rods of Nuclear Reactor Simulator (no date) Nuclearinst.com 6](file:///C:\Users\joshu\Documents\GitHub\Programming-Project\Joshua%20Netherton%20-%20Component%203.docx#_Toc118722666)

[Figure 3 - NUKEMAP by Alex Wellerstein (no date) Nuclearsecrecy.com 7](#_Toc118722667)

[Figure 4 - MH (no date) Nuclear Power Plant Simulator free online game, Github.io. 8](file:///C:\Users\joshu\Documents\GitHub\Programming-Project\Joshua%20Netherton%20-%20Component%203.docx#_Toc118722668)

[Figure 5 – Top-Down Diagram (Figma, Whiteboard and Diagram Maker) 15](#_Toc118722669)

[Figure 6 - Main Program Screen (Flowchart maker & online diagram software, no date) 16](file:///C:\Users\joshu\Documents\GitHub\Programming-Project\Joshua%20Netherton%20-%20Component%203.docx#_Toc118722670)

[Figure 7 – Control Screen (Flowchart maker & online diagram software, no date) 16](file:///C:\Users\joshu\Documents\GitHub\Programming-Project\Joshua%20Netherton%20-%20Component%203.docx#_Toc118722671)

[Figure 8 – Success Screen (Flowchart maker & online diagram software, no date) 17](file:///C:\Users\joshu\Documents\GitHub\Programming-Project\Joshua%20Netherton%20-%20Component%203.docx#_Toc118722672)

[Figure 9 - Failure Screen (Flowchart maker & online diagram software, no date) 17](file:///C:\Users\joshu\Documents\GitHub\Programming-Project\Joshua%20Netherton%20-%20Component%203.docx#_Toc118722673)

[Figure 10 - Revised Main Program Screen (Flowchart maker & online diagram software, no date) 19](#_Toc118722674)

[Figure 11 - Revised Date Selection Window (Flowchart maker & online diagram software, no date) 20](#_Toc118722675)

[Figure 12 - Revised Wind Selection Window (Flowchart maker & online diagram software, no date) 20](#_Toc118722676)

[Figure 13 - Main System Start Up (https://app.diagrams.net/) 21](file:///C:\Users\joshu\Documents\GitHub\Programming-Project\Joshua%20Netherton%20-%20Component%203.docx#_Toc118722677)

[Figure 14 - Internet Check (https://app.diagrams.net/) 22](file:///C:\Users\joshu\Documents\GitHub\Programming-Project\Joshua%20Netherton%20-%20Component%203.docx#_Toc118722678)

[Figure 15 - Online Mode is on (https://app.diagrams.net/) 23](file:///C:\Users\joshu\Documents\GitHub\Programming-Project\Joshua%20Netherton%20-%20Component%203.docx#_Toc118722679)

[Figure 16 - User not connected (https://app.diagrams.net/) 23](file:///C:\Users\joshu\Documents\GitHub\Programming-Project\Joshua%20Netherton%20-%20Component%203.docx#_Toc118722680)

[Figure 17 - Detonate Sequence (https://app.diagrams.net/) 24](file:///C:\Users\joshu\Documents\GitHub\Programming-Project\Joshua%20Netherton%20-%20Component%203.docx#_Toc118722681)

[Figure 18 - Detonation Function (https://app.diagrams.net/) 25](file:///C:\Users\joshu\Documents\GitHub\Programming-Project\Joshua%20Netherton%20-%20Component%203.docx#_Toc118722682)

[Figure 19 - Missing Data (https://app.diagrams.net/) 26](#_Toc118722683)

[Figure 20 - Radius Calculation (https://app.diagrams.net/) 27](#_Toc118722684)

[Figure 21 - Choosing Wind (https://app.diagrams.net/) 28](#_Toc118722685)

[Figure 22 - Date Choose (https://app.diagrams.net/) 29](#_Toc118722686)

[Figure 23 - Setting a marker 36](file:///C:\Users\joshu\Documents\GitHub\Programming-Project\Joshua%20Netherton%20-%20Component%203.docx#_Toc118722687)

[Figure 24 - Defining latitude and longitude 37](file:///C:\Users\joshu\Documents\GitHub\Programming-Project\Joshua%20Netherton%20-%20Component%203.docx#_Toc118722688)

[Figure 25 - Setting polygon points 37](file:///C:\Users\joshu\Documents\GitHub\Programming-Project\Joshua%20Netherton%20-%20Component%203.docx#_Toc118722689)

[Figure 26 - Second page sliders 38](file:///C:\Users\joshu\Documents\GitHub\Programming-Project\Joshua%20Netherton%20-%20Component%203.docx#_Toc118722690)

[Figure 27 - Defining wind choice 38](file:///C:\Users\joshu\Documents\GitHub\Programming-Project\Joshua%20Netherton%20-%20Component%203.docx#_Toc118722691)

[Figure 28 - Defining date choice 38](#_Toc118722692)

[Figure 29 - Setting up main window 39](#_Toc118722693)

[Figure 30 - Frame layout 39](#_Toc118722694)

[Figure 31 - Positioning the map 39](#_Toc118722695)

[Figure 32 - Reactor Selection (app.diagrams.net) 47](file:///C:\Users\joshu\Documents\GitHub\Programming-Project\Joshua%20Netherton%20-%20Component%203.docx#_Toc118722696)

[Figure 33 - Setting data for reactor type (app.diagrams.net) 48](file:///C:\Users\joshu\Documents\GitHub\Programming-Project\Joshua%20Netherton%20-%20Component%203.docx#_Toc118722697)

[Figure 34 - Reset button (app.diagrams.net) 48](file:///C:\Users\joshu\Documents\GitHub\Programming-Project\Joshua%20Netherton%20-%20Component%203.docx#_Toc118722698)

[Figure 35 - Internet ping (app.diagrams.net) 48](file:///C:\Users\joshu\Documents\GitHub\Programming-Project\Joshua%20Netherton%20-%20Component%203.docx#_Toc118722699)

[Figure 36 - Clearing marker and polygons 49](#_Toc118722700)

[Figure 37 - Setting data 50](file:///C:\Users\joshu\Documents\GitHub\Programming-Project\Joshua%20Netherton%20-%20Component%203.docx#_Toc118722701)

# Bibliography

Citations by BibGuru (*BibGuru - A new FREE APA, Harvard, & MLA citation generator* (no date))

* *BibGuru - A new FREE APA, Harvard, & MLA citation generator* (no date) *BibGuru*. Available at: https://www.bibguru.com/ (Accessed: April 28, 2022).
* *IAEA (no date)* *Nuclear reactor simulators for education and training*, *Iaea.org*. Available at: https://www.iaea.org/topics/nuclear-power-reactors/nuclear-reactor-simulators-for-education-and-training (Accessed: May 4, 2022).
* *Creately (https://app.creately.com/d/Fmrhsfl9IiH/edit*). Available at: https://app.creately.com/d/Fmrhsfl9IiH/edit (Accessed: May 5, 2022).
* *Nuclear Reactor Simulator* (no date) *Nuclearinst.com*. Available at: https://www.nuclearinst.com/Nuclear-Reactor-Simulator (Accessed: May 6, 2022).
* *Nuclear Reactor Simulator control rods (no date) Nuclearinst.com*. Available at: https://www.nuclearinst.com/Nuclear-Reactor-Simulator (Accessed: May 13, 2022)
* *NUKEMAP by Alex Wellerstein* (no date) *Nuclearsecrecy.com*. Available at: https://nuclearsecrecy.com/nukemap/ (Accessed: May 13, 2022).
* MH (no date) *Nuclear Power Plant Simulator free online game*, *Github.io*. Available at: https://donovan22.github.io/HTMLTest/NPPS.html (Accessed: May 13, 2022).
* *Linquip Team. (2021) All types of nuclear reactors (PDF & Charts), Industrial Manufacturing* Blog *| linquip. Available at: https://www.linquip.com/blog/types-of-nuclear-reactors/* (Accessed: May 16, 2022)
* *Visual Studio Code - code editing. Redefined* (no date) *Visualstudio.com*. Available at: https://code.visualstudio.com/ (Accessed: 18 May 2022).
* Morgan, W. F. and Sowa, M. B. (2015) ‘Non-targeted effects induced by ionizing radiation: mechanisms and potential impact on radiation induced health effects’, *Cancer letters*, 356(1), pp. 17–21. doi: 10.1016/j.canlet.2013.09.009.
* *Homepage-July16* (no date) *EDF*. Available at: https://www.edfenergy.com/ (Accessed: 26 May 2022).
* *Home* (no date) *Dominionenergy.com*. Available at: https://www.dominionenergy.com/ (Accessed: 26 May 2022).
* *China syndrome* (no date) *Encyclopedia.com*. Available at: https://www.encyclopedia.com/humanities/dictionaries-thesauruses-pictures-and-press-releases/china-syndrome (Accessed: 26 May 2022).
* *Decomposition Diagram* (no date) *mural.co.* Available at: https://app.mural.co/t/dissertationcomponent39445/m/dissertationcomponent39445/1654677648655/2360db62965b3d9e524a2a76e202a49a591dcd3b?sender=u8cb3fc9919c473bad7bc8745
* *Flowchart maker & online diagram software* (no date) *Diagrams.net*. Available at: https://app.diagrams.net/ (Accessed: 23 June 2022).
* *SmartDraw - create flowcharts, floor plans, and other diagrams on any device* (no date) *Smartdraw.com*. Available at: https://www.smartdraw.com/?id=104640 (Accessed: 30 June 2022).
* *Gestalt Principles (no date) interaction-design.org.* Available at: https://www.interaction-design.org/literature/topics/gestalt-principles (Accessed: 26 July 2022).
* *Figma, whiteboard and diagram website* (no date) *figma.com.* Available at: https://www.figma.com/file/nbYDdYkAGlcqYUxEy0m79T/Untitled?node-id=0%3A1 (Accessed: 08 September 2022)
* *Draw Diagrams* (no date) *draw.io.* Available at: https://app.diagrams.net/ (Accessed: 23 September 2022)
* Schimansky, T. (no date) *CustomTkinter: A modern and customizable python UI-library based on Tkinter*.
* *tkinter — Python interface to Tcl/Tk — Python 3.10.7 documentation* (no date) *Python.org*. Available at: https://docs.python.org/3/library/tkinter.html (Accessed: September 29, 2022).
* Schimansky, T. (no date) *TkinterMapView: A python Tkinter widget to display tile-based maps like OpenStreetMap or Google Satellite Images*.
* *PySimpleGUI* (no date) *Pysimplegui.org*. Available at: https://www.pysimplegui.org/en/latest/ (Accessed: September 29, 2022).
* *tkcalendar — tkcalendar 1.5.0 documentation* (no date) *Readthedocs.io*. Available at: https://tkcalendar.readthedocs.io/en/stable/ (Accessed: September 29, 2022).
* *Bfs – Environmental Impact of Fukushima meltdown* (no date) *www.bfs.de.* Available at: https://www.bfs.de/EN/topics/ion/accident-management/emergency/fukushima/environmental-consequences.html#:~:text=Following%20the%20reactor%20accident%20in,the%20high%20levels%20of%20radiation. (Accessed: November 7, 2022)
* *Cumbria Government – Damage of Sellafield Reactor* (no date) *cumbria.gov.uk*. Available at: https://cumbria.gov.uk/elibrary/Content/Internet/535/600/41458111550.PDF (Accessed: November 7, 2022)