Tin Whiskers

Unity 3D App

National Aeronautics and Space Administration

Missile Defense Agency

Naval Surface Warfare Center

Tin Whisker Investigative Team

Trevor Buchanan, Gavin Mockett, Jonathan Sun

Project Repo: https://github.com/WSUCptSCapstone-S24-F24/-mda-unity3dapp-

Table of Contents

I. Introduction

In May of 1998, only 5 years into its 15+ year mission, the commercial communications satellite known as GALAXY IV suffered an abrupt and terminal failure rendering it inoperable [2]. It provided services like processing financial transactions, distributing syndicated radio shows, and pager services. When the GALAXY IV satellite failed without warning and without immediate backup available, all of its services were out of operation for a period of 24 to 48 hours. At that time, NASA was about to accept the delivery of a “very similarly made” satellite from the maker of the GALAXY IV satellite. However, before accepting the potentially afflicted new satellite, they were invited by the supplier to review their investigation into the root cause of the failure. The failure was discovered to be metallic crystal formations known as “whiskers”. These whiskers can form in almost all environments, on many different metals, provided certain select conditions are met.

It is this project's goal to develop a software tool that can capture a 3D model of a printed circuit board (PCB), identify its exposed conductors, and simulate a storm of detached metal whiskers landing on the PCB. This program will be a continuation of the prior students’ project that employed the Unity (Physics) game engine for development. This project does not involve research into the mechanism(s) for whisker nucleation and growth, nor the mechanism(s) for their detachment.

The project is situated in the domain of electronics reliability and 3D modeling, specifically addressing the issue of tin whiskers in electronic components. Tin whiskers are electrically conductive hair-like structures that can grow from the surface of tin coatings [1], potentially causing short circuits and other electrical failures.

There have been attempts in the past to simulate the effects of tin whiskers and to find the probability of problems occurring. One attempt to quantify the reliability risk of tin whisker failures is from the Center for Advanced Life Cycle Engineering (CALCE) [3]. CALCE developed a spreadsheet-based prediction software system to calculate the expected number of tin whisker shorts on one or more leaded components, based on user-specified part and lead geometry, whisker density, tin whisker length distribution, applied voltage, and conformal coating coverage. However, the system developed by CALCE is not considered to be a cutting-edge model. The model has proven to be overly pessimistic in application.

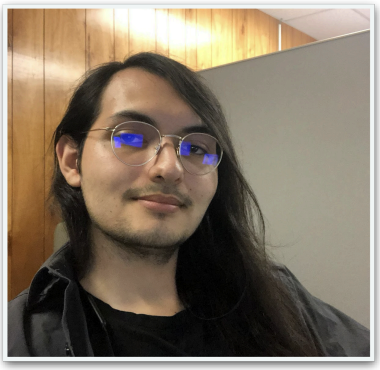
Another attempt to quantify the reliability risks from tin whiskers is the Raytheon Analysis Laboratory algorithm by David Pinsky [5]. This model ranks the potential for whisker growth on a scale of 1 to 10, where 1 is a very low risk and 10 is a very high risk for whiskers to bridge connections at different potentials.

II. Team Member & Bios

Trevor Buchanan

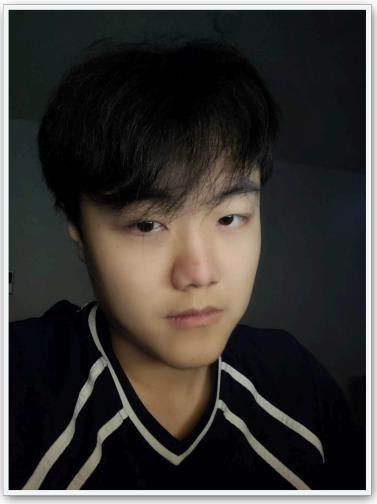
Trevor Buchanan is a Computer Science student at Washington State University, poised to graduate in 2024 with a minor in Mathematics. His technical skill set includes programming languages such as C, C++, Python, and Java, alongside expertise in Agile, MVVM, and MVC frameworks. A graduate of Tacoma Community College with an Associate of Arts, Trevor has honed his problem-solving and adaptability skills through rigorous coursework in Cyber Security, Cryptography, and Software Design. His interests span from coding and mathematics to more creative pursuits like art and woodworking, reflecting a well-rounded personality capable of blending analytical rigor with creative thinking.

Gavin Mockett



Gavin Mockett is a dedicated Computer Science student at Washington State University, set to graduate in 2024, with comprehensive technical skills in programming languages like C, C++, Python, and Lua, as well as in Unity development and cybersecurity. His expertise extends to Ansible automation and Redhat administration, developed through a cybersecurity internship at the Pacific Missile Facility Range. As the team coordinator for WSU's Esports Club, Gavin has honed his leadership and project management skills, consistently training over 45 students since 2021. His intellectual pursuits include exploring advanced mathematical concepts and the integration of computing with music and art, showcasing his diverse interests and continuous desire to learn.

Jonathan Qiaoshun



Jonathan Qiaoshun is a Computer Science major at Washington State University, anticipated to graduate in 2024, with strong interests in video game and AI development. Currently an intern at AIA Information Technology Co., Ltd, he develops apps and manages data using SharePoint and Power Automate, alongside testing and enhancing software functionalities. Previously, as a TA for CPTS 223 at WSU, he handled coursework evaluation and student records. Fluent in English and Chinese, Jonathan's technical skills include Java, C/C++, and proficiency in Microsoft Office, backed by solid capabilities in project management and software design principles.

III. Requirements and Specifications

III.1. Project Stakeholders

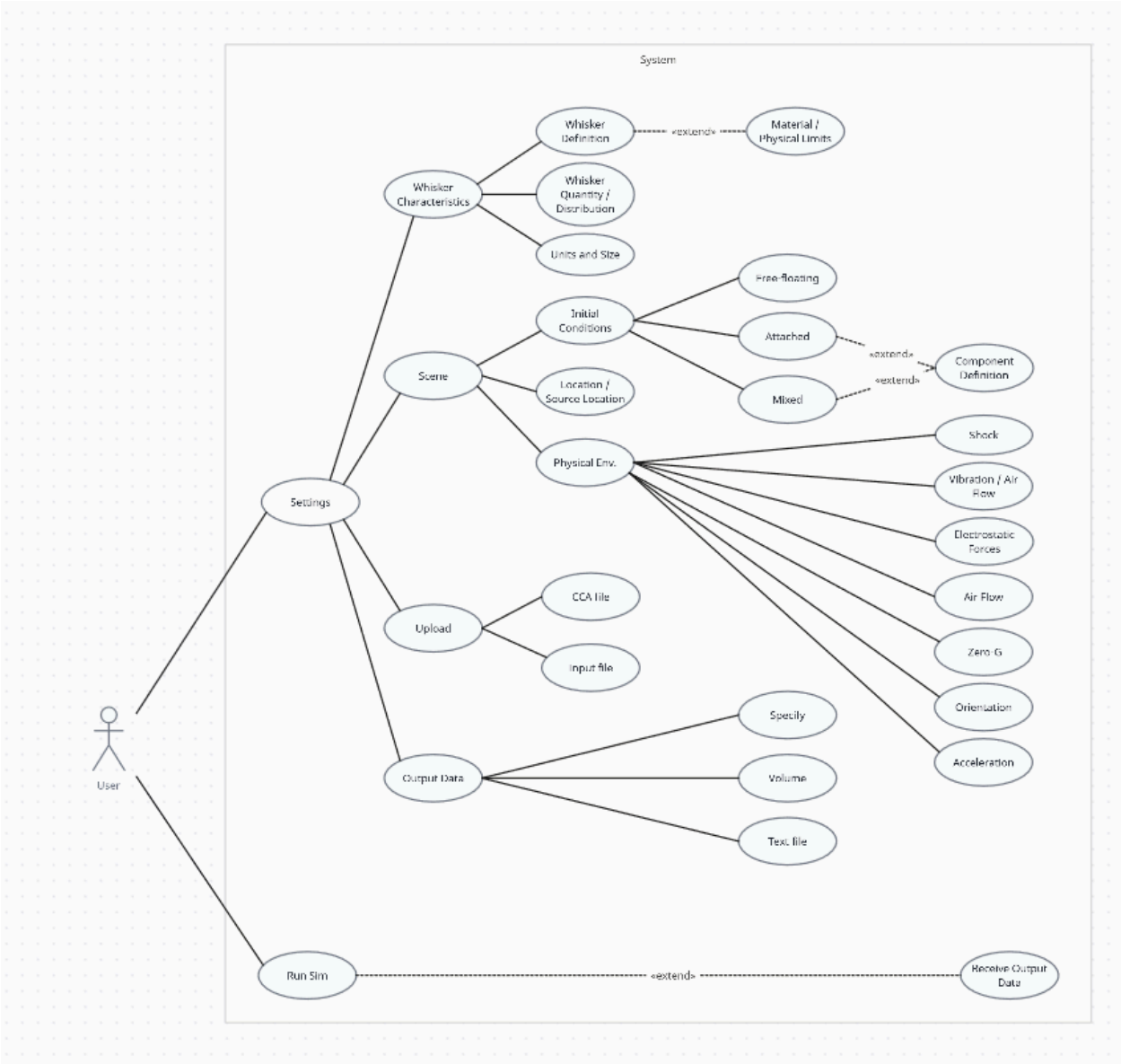
The three primary clients of this project are the National Aeronautics and Space Administration (NASA), the Missile Defense Agency (MDA), and the Naval Surface Warfare Center. However, because tin whiskers (and other metal whiskers) pose problems in so many fields, the final product can be applied by many other organizations.

Any organization that uses circuitry can apply this program to better understand the risks of tin whiskers and to prevent failures caused by them. Potential clients for this project include any corporation that stands to benefit from this product.

Government stakeholders require that our application prioritize safety and security.

Stakeholders in this project will benefit from the results that this program can provide. The project is also open-source. This allows easy access and use for all interested parties. Our team will ensure that the progress is well documented and the code is efficient, extendable, and reusable. The needs of our primary clients will be considered first in the development process, but the needs of other interested parties will be considered as well.

III.2. Use Cases



Upload CCA file

Pre-condition	- On the main menu
Post-condition	- New CCA is initialized and configured
Basic Path	1. Locate Settings 2. Navigate to Upload 3. The user uploads their target PCB and associated components using a OBJ file

Alternative Path	
Related Requirements	- User needs OBJ file - Main Menu

Enable Vibration

Pre-condition	- On the main menu
Post-condition	- The simulation environment will have vibrations specified by the user
Basic Path	1. Locate Settings 2. Navigate to Scene 3. Navigate to Physical env. 4. Enable Vibration 5. Specify vibration settings
Alternative Path	
Related Requirements	- Main Menu

Edit Whisker Material:

Pre-condition	- On the main menu
Post-condition	- User will have selected the preferred type of whisker material to be simulated
Basic Path	1. Locate Settings 2. Navigate to Whisker Characteristics 3. Navigate to Whisker Definition 4. User selects preferred material type 5. User enters in parameters of the whisker itself (i.e., # of detached whiskers, whisker length, and whisker thickness distribution parameters)
Alternative Path	
Related Requirements	- Main Menu

Run Simulation

Pre-condition	- On the main menu
Post-condition	- Receive Data Output
Basic Path	1. Configure preferred settings 2. Navigate back to the main menu 3. Run Simulation 4. Collect Output
Alternative Path	
Related Requirements	- Settings are properly configured for preferred simulation - Main Menu - Working Simulation

Physical Limits:

Description	User to enter variables that define the number of detached whiskers as well as the material composition of the whiskers, the statistical distribution parameters (mu, sigma) for whisker length and whisker thickness to use when simulating the population of detached whiskers
Source	Requirement specified by primary clients (NASA, MDA, NAVSEA)
Priority	Priority Level 0: Essential and required functionality

III.3. Functional Requirements

1. Inputs

Initial Conditions:

Description	Spawns detached whiskers into the initial scene from a bounding box/volume at specified location in proximity to the target PCB
Source	Requirement specified by primary clients (NASA, MDA, NAVSEA)
Priority	Priority Level 0: Essential and required functionality

Printed Circuit Board (PCB) and Associated Components Definition:

Description	The product's physical components must abide by the bounds that the whiskers will be subjected to
Source	Requirement specified by primary clients (NASA, MDA, NAVSEA)
Priority	Priority Level 0: Essential and required functionality

Physical Environment:

Description	The project simulation's tin whiskers must be affected by the state of the environment that they are placed in; such as: - Shock (0) - Orientation (0) - Gravity/ Force directions /Zero-G (0) - Vibration (1) - Electrostatic forces (2) - Air flow (2)
Source	Requirement specified by primary clients (NASA, MDA, NAVSEA)
Priority	Priority Level Mixed

Whisker Acceleration:

Description	The whiskers inside of the product's simulation must act according to specified acceleration setting (constant velocity, accelerating, etc.)
Source	Requirement specified by primary clients (NASA, MDA, NAVSEA)
Priority	Priority Level 0: Essential and required functionality

2. System Level Function

Geometric Surfaces:

Description	The product must allow the description of the physical surface of the node, such as the shape.
Source	Requirement specified by primary clients (NASA, MDA, NAVSEA)
Priority	Priority Level 0: Essential and required functionality

System Container/Bounding Box:

Description	All of the functionalities of the application, including whisker generation and whisker dispersal, must function inside of the prescribed container/bounding box
Source	Requirement specified by primary clients (NASA, MDA, NAVSEA)
Priority	Priority Level 0: Essential and required functionality

Coordinates:

Description	Within the simulation, the x, y, z coordinates of an must be attached to the ID number in order to identify it
Source	Requirement specified by primary clients (NASA, MDA, NAVSEA)
Priority	Priority Level 0: Essential and required functionality

Create ID Number:

Description	In the project's simulation, each metal whisker generated must be assigned to a unique ID number.
Source	Requirement specified by primary clients (NASA, MDA, NAVSEA)
Priority	Priority Level 0: Essential and required functionality

Tag Database:

Description	The product must have a database that stores the following: - Whisker ID number and location - Node pair ID number and location: Data from identifying and labeling node pairs
Source	Requirement specified by primary clients (NASA, MDA, NAVSEA)
Priority	Priority Level 0: Essential and required functionality

3. Simulation

Store Bridges' IDs:

Description	Simulation must identify bridged nodes ID and whisker ID will be stored.
Source	Requirement specified by primary clients (NASA, MDA, NAVSEA)
Priority	Priority Level 0: Essential and required functionality

Identify Surface Geometry:

Description	The applications must identify the surface geometry of a node where the whisker touches the electronic component: What the bridged node geometry looks like, as extracted from the identification database
Source	Requirement specified by primary clients (NASA, MDA, NAVSEA)
Priority	Priority Level 0: Essential and required functionality

Whisker to Whisker

Description	During the application's simulations, any whisker in contact with another whisker must identify the additional whisker.
Source	Requirement specified by primary clients (NASA, MDA, NAVSEA)
Priority	Priority Level 2: Extra features or stretch goals

Coordinates Log:

Description	During the application's simulations, if a whisker comes in contact with (exposed) electrical conductive surface/nodes, then the location of contact (the coordinate where the contact occurred) must be logged.
Source	Requirement specified by primary clients (NASA, MDA, NAVSEA)
Priority	Priority Level 0: Essential and required functionality

Chain Connection:

Description	After the application's simulations, the number and ID of the whisker(s) in each connection/chain connection for each short must be recorded
Source	Requirement specified by primary clients (NASA, MDA, NAVSEA)
Priority	Priority Level 2: Extra features or stretch goals

4. Outputs

Saving:

Description	After running a simulation, all necessary data (inputs, whisker connection IDs, etc) must be saved in the identification database, contact database, and short database.
Source	Requirement specified by primary clients (NASA, MDA, NAVSEA)
Priority	Priority Level 0: Essential and required functionality

5. Graphical User Interface (GUI)

Screen Capture:

Description	The product must have the ability to take screenshots or recordings of the circuit, including when the tin whiskers may be moving around.
Source	Requirement specified by primary clients (NASA, MDA, NAVSEA)
Priority	Priority Level 1: Desirable functionality.

Highlight:

Description	Distinction of shorted components that are easily identifiable for the user visually
Source	Requirement specified by primary clients (NASA, MDA, NAVSEA)
Priority	Priority Level 1: Desirable functionality

Edit Circuit:

Description	Add components, delete components, and modify the original circuit
Source	Requirement specified by primary clients (NASA, MDA, NAVSEA)
Priority	Priority Level 1: Desirable functionality

Node identification:

Description	Allows users to view and modify the node identifications from the identification database.
Source	Requirement specified by primary clients (NASA, MDA, NAVSEA)
Priority	Priority Level 1: Desirable functionality

III.4. Non-Functional Requirements

Performance:

The time it takes for the system to respond to a user request should be minimal. The system should have the ability to handle increased load or data volume. The system should continue operating in the event of a failure.

Compatibility:

The project should have the ability of the system to work seamlessly with other systems or components. Ensuring the system works correctly with different browsers, devices, or operating systems.

User Interface (UI) Design:

The system's interface should be user-friendly and intuitive. It should be easy to use and clear. Data should be labeled and displayed in an organized style. Users should be able to interpret the data clearly.

Maintainability:

The project should have the ease with which the system can be changed or extended because this project covers a lot of data and functions and will be consistently used in the future. Methods and functions are expected to be commented accordingly

Scalability:

The project should have the ability to handle future growth or changes in requirements. This is an ongoing project with room to expand.

Documentation:

Clear and comprehensive documentation for code, architecture, references, and system components.

IV. Software Design - From Solution Approach

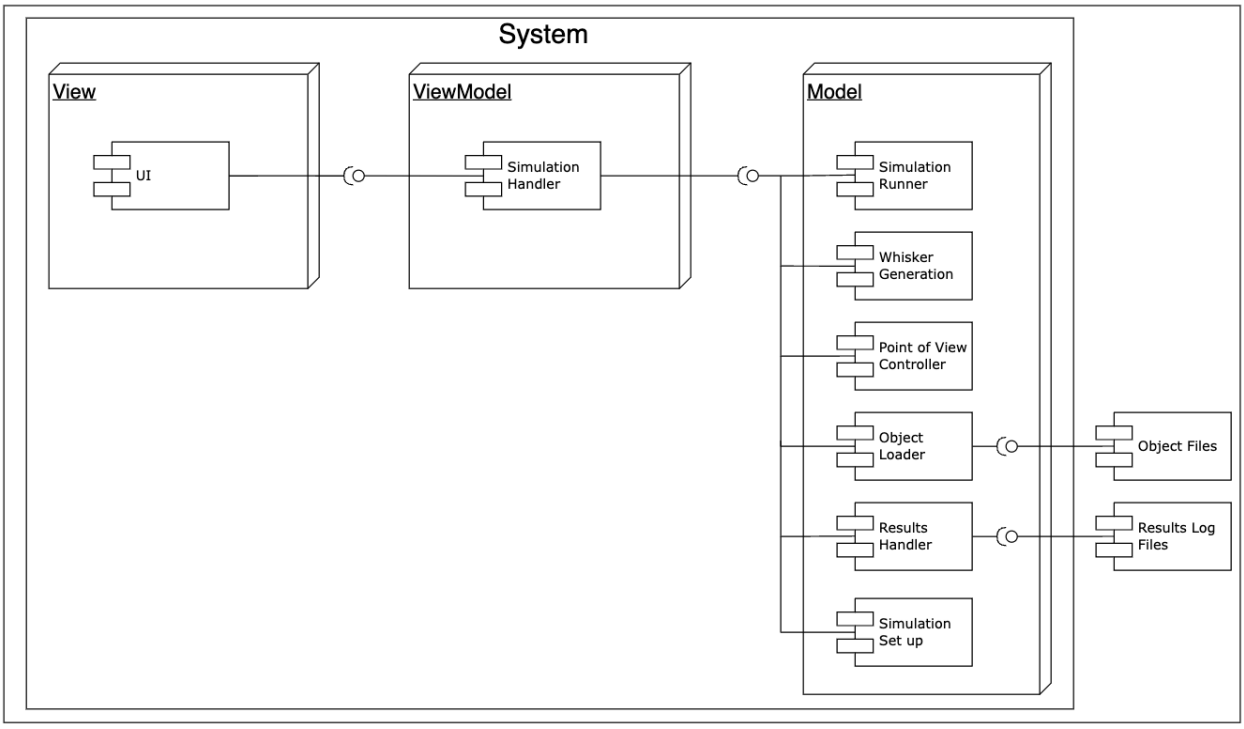
IV.1.hArchitecture Design

IV.1.1. Overview

The software architecture of our tin whisker simulation tool follows a modular and layered approach, with a focus on scalability, maintainability, and flexibility. We have adopted the Model-View-ViewModel (MVVM) architectural pattern due to its suitability for separating concerns and facilitating modular development.

- Model: Represents the core data and logic of the application, including PCB models, conductor data, whisker characteristics, and simulation results. Separating this data from the user interface enhances reusability and testability.
- View: Presents the data to the user and handles user input. This layer includes the graphical user interface components responsible for displaying PCB models, configuring simulations, and visualizing results. The view gets updated through data-binding with the ViewModel.
- ViewModel: Acts as an intermediary between the model and the view, coordinating user interactions and updating the model accordingly. This layer contains the business logic and uses the model's logic to get results to send back to the view layer.

Because we are utilizing the Unity engine, we are using the architectural pattern that we feel is the most natural for Unity. Unity takes input from the user (View), sends it to be processed and handled (ViewModel) to delegate to the engine/logic (Model), and then updates the view accordingly. Below is a broad overview of the structure of the planned software architecture for our project.



IV.1.2. System Decomposition

Each of the components in each block described above can be approached individually. This way our team can approach the project in a block-based manner. Each of the members of our team can implement components without collisions or pipeline progress issues. The only potential caveat is the components cannot be correctly tested and analyzed unless the other components are implemented, although they can still run without the implementation of the rest. Since we are working with Unity, the components are broken into sections that can be scenes or tasks. In Unity, scenes are where you work with content. They are assets that contain all or part of a game or application. This way, the system is designed to achieve high cohesion throughout subsystems while minimizing coupling between them. Each subsystem corresponds to a distinct area of functionality, allowing for independent development and testing. Interfaces between subsystems are carefully defined to ensure clear communication and minimize dependencies.

2.1. UI

Description:

The UI controls all of the user-side information and displays. This includes getting user inputs and updating the display as specified.

Concepts and Algorithms Generated:

The built-in UI in Unity controls most of the complex algorithms concerning displays and user input. However, this component will still be split into subclasses that perform the tasks of the UI component.

Interface Description:

Services Provided:

Service Name	Service Provided To	Description
Handle Input	Simulation Handler	Once the user inputs are received, the UI component is meant to send necessary commands and directions to the Simulation Handler.
Display Output	Unity UI	After the inputs have been processed, send the updates to display to Unity UI.

Services Required:

Service Name	Service Provided From
Receive Input	Unity UI

2.2. Simulation Handler

Description:

The simulation handler component is meant to be the main part of the ViewModel layer. It receives the directions from the UI (View) and sends them to be processed in the Model layer. Then it sends the results from the Model layer back to the UI (View) layer to be handled and displayed.

Concepts and Algorithms Generated:

This component is the interface between the View layer and the Model layer. It controls directing all of the Model components and sending the necessary information back to the View layer to be processed and displayed (through data binding).

Interface Description:

Services Provided:

Service Name	Service Provided To	Description
Handle Tasks	UI	Call and control all of the components in the Model layer as specified.
User Inputs	Point of View Handler, Simulation Set up, Results Handler	Gives the necessary user inputs to components as needed.
Collection of Whiskers	Simulation Runner	Gives the gathered collection of whispers from the Whisker Generator.
Simulation Settings	Simulation Runner	Provides the simulation settings for the simulation runner to use in its simulation.
Results from Simulation	Results Handler	Gives the results gathered from a simulation to the results handler.

Services Required:

Service Name	Service Provided From
User Inputs	UI
Camera orientation and position updates	Point of View Handler
Collection of Whiskers	Whisker Generation
Simulation Settings	Simulation Set up
Results from Simulation	Simulation Runner

2.3. Simulation Runner

Description:

Perform a simulation of tin whiskers falling as specified.

Concepts and Algorithms Generated:

This component contains logic for running a simulation (whiskers on a PCB) and collecting and recording data throughout the simulation. This will require collision detection algorithms to find when and where nodes are connected by a metal whisker.

Interface Description:

Services Provided:

Service Name	Service Provided To	Description
Records simulation data	Simulation Handler	Gathers and records node pairs and their data

Services Required:

Service Name	Service Provided From
Collection of Generated Whiskers	Simulation Handler

2.4. Whisker Generation

Description:

The Whisker Generation component controls the creation of all of the metal whiskers.

Concepts and Algorithms Generated:

This component uses the content in the Simulation Set up component to create a collection of metal whiskers as specified by the user.

Interface Description:

Services Provided:

Service Name	Service Provided To	Description
Whisker creation	Simulation Handler	Creates a collection of whiskers as specified by the user.

Services Required:

Service Name	Service Provided From
Whisker settings	Simulation Handler

2.5. Point of View Controller

Description:

Orients the camera to view what the user requests.

Concepts and Algorithms Generated:

The matrix manipulation and other mathematical concepts used in this component will be handled by the Unity Engine. The Unity Engine has extensive tools dealing with view handling.

Interface Description:

Services Provided:

Service Name	Service Provided To	Description
Camera orientation and position updates	Simulation Handler	Sends camera orientation and position updates to the simulation handler

Services Required:

Service Name	Service Provided From
User Inputs	Simulation Handler

2.6. Object Loader

Description:

The Object Loader component handles the logic for loading in a PCB for a simulation.

Concepts and Algorithms Generated:

The file search and system requirements will be met by Unity's built-in file searching tools.

Interface Description:

Services Provided:

Service Name	Service Provided To	Description
Loaded PCB or other object file	Simulation Handler	Allows the user to search their files and select a PCB object file to load into the simulation, and then load it in.

Services Required:

Service Name	Service Provided From
File Search	Unity Engine
User Input	Simulation Handler
Object file (of PCB)	Object Files

2.7. Results Handler

Description:

The Results Handler component handles the logic for analyzing inputs and writing them to the necessary log files.

Concepts and Algorithms Generated:

This component uses Python to generate a heat map of the node-pairs generated during a simulation run.

Interface Description:

Services Provided:

Service Name	Service Provided To	Description
Results log	Results Log Files	Writes results to log files.

Services Required:

Service Name	Service Provided From
Heat map generator	Python code

2.8. Simulation Set up

Description:

The simulation set up component holds all the information needed to run a simulation. This includes fields such as whisker count, whisker size, object positions, and so on.

Concepts and Algorithms Generated:

This component receives directions from the simulation handler and sets up a simulation accordingly.

Interface Description:

Services Provided:

Service Name	Service Provided To	Description
Set Simulation Content	Simulation Handler	This service sets up a simulation with specified inputs by setting the necessary values of a simulation's content.

Services Required:

Service Name	Service Provided From
Content Inputs	Simulation Handler

IV.2. Data Design

Data Structures:

1. Printed Circuit Board (PCB) Model:
    - The PCB model represents the physical layout of the printed circuit board, including its components, traces, and conductive paths.
    - Data structure: 3D mesh or graph representation, storing node and edge information to define the PCB topology.
2. Conductor Identification Data:
    - Data structure storing information about the location and properties of exposed conductors on the PCB surface.

- Data structure: Array, list, or dictionary storing coordinates, conductivity, and other relevant attributes of exposed conductors.

3. Whisker Characteristics:
   - Data structure to define the characteristics of detached metal whiskers, such as material composition, length, thickness, and distribution.
   - Data structure: Object or dictionary storing whisker properties as key-value pairs.
4. Simulation Parameters:
   - Data structure to store user-defined simulation parameters, including the number of whiskers, simulation duration, and environmental conditions.
   - Data structure: Object or dictionary storing simulation settings as key-value pairs.
5. Simulation Results:
   - Data structure to store the results of whisker simulation, including information about bridged conductor pairs and probability of failures.
   - Data structure: Array, list, or database table storing simulation output for analysis and visualization.

Databases:

1. Simulation Data Repository:
   - Database to store simulation input and output data for future reference and analysis.
   - Tables:
     i. PCB models: Store information about PCB geometries and components.
     ii. Conductor data: Store details of exposed conductors identified in PCB models.
     iii. Whisker characteristics: Store parameters defining whisker properties.
     iv. Simulation settings: Store user-defined simulation parameters.
     v. Simulation results: Store output data from whisker simulation, including bridged conductor pairs and failure probabilities.

2. User Settings Database:
   - Database to store user preferences and settings for customizing the software tool's behavior.
   - Tables:
     i. User profiles: Store user information and preferences.
     ii. Default settings: Store default values for simulation parameters and tool configurations.
     iii. These data structures and databases will serve as the foundation for storing and managing the information required for modeling, simulating, and analyzing tin whisker-induced failures in electronic components. Properly understanding and implementing these data structures and databases are crucial for ensuring the effectiveness and scalability of the software application.

IV.3. User Interface Design

The user interface (UI) for the tin whisker simulation software will be designed with a focus on intuitiveness, ease of use, and visual clarity. It will consist of several key components to facilitate user interaction with the software tool. Below is a detailed description of each component, accompanied by mock-up images:

1. Main Dashboard:
    - The main dashboard serves as the entry point for users and provides access to various functionalities of the software tool.
    - Use cases: Launching simulations, accessing PCB modeling tools, viewing simulation results.

2. PCB Modeling Tool:
    - This tool allows users to create and modify 3D models of printed circuit boards (PCBs) by adding components, traces, and conductive paths.
    - Use cases: Creating new PCB models, and editing existing PCB layouts.

3. Conductor Identification Interface:
    - Users can use this interface to identify and mark exposed conductors on the PCB surface, either manually or through automated detection algorithms.
    - Use cases: Identifying exposed conductors for whisker simulation, and adjusting conductor properties.

4. Whisker Simulation Configuration:
    - This interface allows users to specify simulation parameters such as whisker material, density, length distribution, and environmental conditions.
    - Use cases: Setting up whisker simulation scenarios, and defining simulation parameters.

5. Simulation Results Viewer:
    - Users can visualize and analyze the results of whisker simulations, including bridged conductor pairs and the probability of failures.
    - Use cases: Analyzing simulation outcomes, and identifying potential failure points.

6. Settings and Preferences:
    - Users can customize the software tool's behavior and appearance through settings and preferences.
    - Use cases: Adjusting default simulation parameters, and changing UI themes.

7. Help and Documentation:
    - This section provides users with access to user manuals, tutorials, and technical documentation to assist them in using the software effectively.
    - Use cases: Accessing help resources, and seeking assistance with software functionalities.

These interface components will enable users to interact with the software tool seamlessly, from creating PCB models to simulating whisker events and analyzing simulation results. The design prioritizes clarity, simplicity, and functionality to ensure a positive user experience and effective utilization of the software's capabilities.

V. Test Case Specification and Results

V.1. Testing Overview

Project Overview

The tin whiskers simulation is a unity simulation being made in collaboration with the listed stakeholders, its purpose is to simulate metallic whiskers falling onto a modeled PCB, find different connections between nodes on the PCB made by said whiskers, and log the data for future use. The program itself should be organized for ease of access, with the biggest focus being the pliability of different variables. The simulation using the Unity Engine poses different challenges, it is not straightforward to test the scripts by themselves as they have to be linked to real-time scenes. This in turn means if you want to test one part of the simulation the entire simulation must be functional, this in turn forces the focus of the project to assure quality in modulation of components.

Test Objectives and Schedule

The objectives of tests for this project are to assure confidence that the scripts are running successfully and consistently to their desired behavior, and beyond that check that all the scenes are properly loaded and using said scripts. Currently, the project is staged with manual tests as many of the changes are focused on retaining the current functionality while making improvements in modularity. Once this stage is proven, having clients use the project itself will be the next objective which will revolve around integrating new features and improving the GUI.

As this is a Unity project, unit testing is more difficult as the entire program is run every time, instead of using a test bed like Unity Test Framework we are focusing on manual testing as most of the additions to the programs use packages and add-ons, such as the monte carlo simulation package. Once again with it being a Unity project integration testing is effectively happening every time the project is run, this process will be heavily used when integrating in systems that are created entirely by us. With most all tests being manual, there will be documentation of what is being tested and how.

The processes focus on a few deliverables. Script testing, Scene testing, and UI testing. All components added to the project are considered non-trivial thus the deliverables focus on the level of abstraction of the engine itself. Script testing focuses on the C# scripts code, the modularity, the effectiveness, and the efficiency. Scene testing focuses on Unity-based scenes, in how the components are connected, what variables they pull from the scripts, and what packages are being activated at what times. Lastly, a subsection of Scene testing is UI testing,

which requires user feedback, how the client wants to access the variables, how much control they want over the program, and how usable the many parts of the program are.

Scope

This section focuses on how we plan to test and create tests for the whiskers simulation. It covers the general process as well as a few examples of the types of tests being used but does not cover all of the tests being held nor the specific inputs for those tests.

V.2. Environment Requirements

Physical Characteristics: Standard development hardware (computers, monitors), Unity development environment.

Communications: Internet access for collaboration and remote testing.

System Software: Unity 3D Engine, version control system (e.g., Git).

Special Test Tools: Testing frameworks (e.g., NUnit for unit testing), and simulation hardware for performance testing.

V.3. Testing Results

As mentioned, the tests implemented by our team were manual assurance checks. These high-level user-managed system tests ensured that the system worked under normal use cases and with a few edge cases. Future teams could greatly improve the testing with unit, integration, and system-level tests to improve stability, reliability, and consistent accuracy. At the end of the project, the integrated Unity testing system *Unity Test Framework* was added to the project to make the process of testing the application easier for future developers. Aside from the technical testing, two major forms of assurance metrics were performed on our project's code base. The first was a static code analysis and the second was a manual checklist of STIGs (for security). Both of these can be found on the project GitHub.

VI. Projects and Tools Used

The software prototype offers a specialized simulation environment for the analysis of whisker growth on PCBs (Printed Circuit Boards). The system architecture is structured into three primary components that align with the MVVM design pattern:

- **View**: This layer comprises the UI elements that the end-users interact with. It presents the simulation data and results in a user-friendly manner and facilitates user inputs and commands that control the simulation process.
- **ViewModel**: Serving as an intermediary between the View and the Model, the ViewModel contains the Simulation Handler, which processes user inputs from the View, translates them into actions or data manipulations in the Model, and maintains the state of the simulation.
- **Model**: This core component is responsible for executing the simulation logic and includes several key modules:

1. PCB Modeling Subsystem

1.1. Functions and Interfaces Implemented

Implemented functionality to create a basic 3D model of a PCB using Unity's Objects and transforms.

Started implementing user interface components for configuring PCB dimensions and adding components.

The remaining work includes refining the user interface, adding more detailed component options, and integrating with the simulation subsystem.

1.2. Preliminary Tests

Conducted unit tests on PCB model creation and component addition functions.

Integration tests with the simulation subsystem are pending until both subsystems are further developed.

2. Whisker Simulation Subsystem

1.1. Functions and Interfaces Implemented

Implemented whisker spawning and physics interactions using Unity's particle system and Rigidbody components.

Integrated basic whisker behavior such as falling, collision detection, and potential bridging between conductors.

User interface components for configuring whisker parameters are partially implemented.

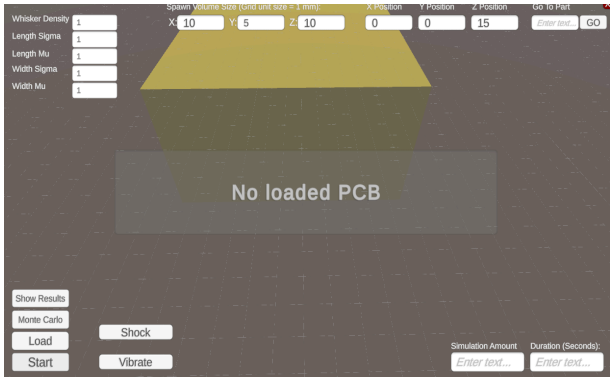
1.2. Preliminary Tests

Conducted unit tests on whisker spawning and physics interactions.

Integration tests with the PCB modeling subsystem are pending for a more comprehensive simulation.

3. GUI Subsystem

1.1. Functions and Interfaces Implemented



The current iteration of the GUI uses the Unity UI and Engine. It serves to collect the user parameters, and then once the simulation is run it houses the results for user access inside the program. Another major addition to the GUI is the popup system, where the user can be notified on what step might be missing to run the simulation properly and to give feedback on when a simulation is run and ends.

1.2. Preliminary Tests

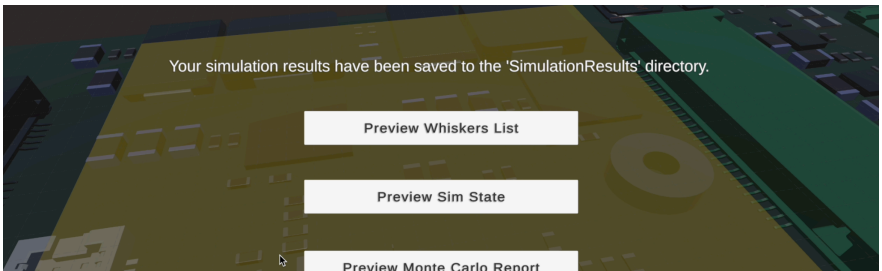
Conducted unit tests on proper UI locking, the locking prevents the regular simulation from being run multiple times so that we can prevent scenes from building up and conflicting.

Integration tests were run for the popup system which has its own queuing system so the pop ups objects don’t build up and conflict. These tests also include running through the singleton format so the popups can be used in any script regardless of location.

4. Results Handler

1.1. Functions and Interfaces Implemented

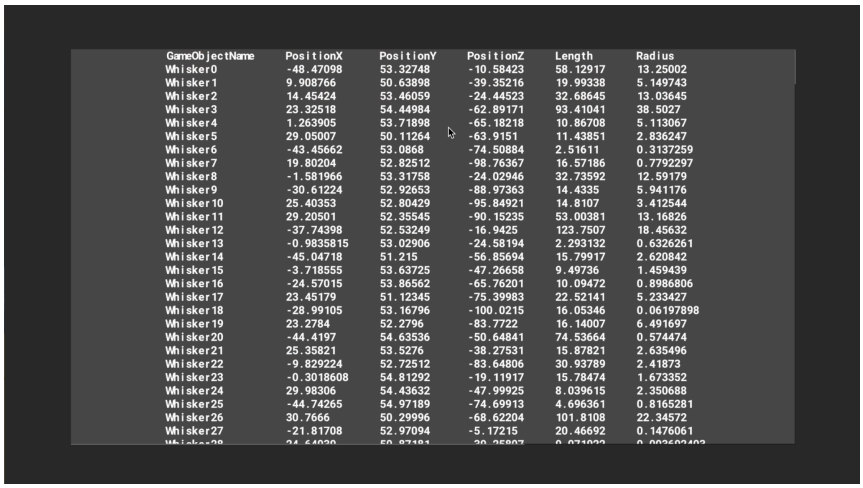
The previous approach of getting



results from a simulation involved generating results via a Python-based heat map, which presented several drawbacks. Primarily, users were unable to simply execute the simulation and obtain the results. Instead, they had to install Python, set up a virtual environment to install required libraries, and then run the program either in a dedicated IDE or via the terminal/command prompt. This process was not user-friendly.

To address this, the results handler component has been revamped to write the results to CSV files. These files store logs for each simulation's results individually. Users can now preview each simulation's results within the application or save them to their device by accessing the saved files in the 'SimulationResults' directory.

The results handler component encompasses functions that write lists of all whiskers and their associated data, the simulation state and its data, the Monte Carlo results and their data, and a log of all bridged component trios (whisker, component1, and component2).



GameObjectName	PositionX	PositionY	PositionZ	Length	Radius
Whisker0	-48.47098	53.32748	-10.58423	58.12917	13.25002
Whisker1	9.908766	50.63898	-39.35216	19.99338	5.149743
Whisker2	14.45424	53.46059	-24.44523	32.68645	13.03645
Whisker3	23.32518	54.44984	-62.89171	93.41041	38.5027
Whisker4	1.263905	53.71898	-65.18218	10.86708	5.113067
Whisker5	29.05007	50.11264	-63.9151	11.43851	2.836247
Whisker6	-43.45662	53.0868	-74.50884	2.51611	0.3137259
Whisker7	19.80204	52.82512	-98.76367	16.57186	0.7792297
Whisker8	-1.581966	53.31758	-24.02946	32.73592	12.59179
Whisker9	-30.61224	52.92653	-88.97363	14.4335	5.941176
Whisker10	25.40353	52.80429	-95.84921	14.8107	3.412544
Whisker11	29.20501	52.35545	-90.15235	53.00381	13.16826
Whisker12	-37.74398	52.53249	-16.9425	123.7507	18.45632
Whisker13	-0.9835815	53.02906	-24.58194	2.293132	0.6326261
Whisker14	-45.04718	51.215	-56.85694	15.79917	2.620842
Whisker15	-3.718555	53.63725	-47.26658	9.49736	1.459439
Whisker16	-24.57015	53.86562	-65.76201	10.09472	0.8986806
Whisker17	23.45179	51.12345	-75.39983	22.52141	5.233427
Whisker18	-28.99105	53.16796	-100.0215	16.05346	0.06197898
Whisker19	23.2784	52.2796	-83.7722	16.14007	6.491697
Whisker20	-44.4197	54.63536	-50.64841	74.53664	0.574474
Whisker21	25.35821	53.5276	-38.27531	15.87821	2.635496
Whisker22	-9.829224	52.72512	-83.64806	30.93789	2.41873
Whisker23	-0.3018608	54.81292	-19.11917	15.78474	1.673352
Whisker24	29.98306	54.43632	-47.99925	8.039615	2.350688
Whisker25	-44.74265	54.97189	-74.69913	4.696361	0.8165281
Whisker26	30.7666	50.29996	-68.62204	101.8108	22.34572
Whisker27	-21.81708	52.97094	-5.17215	20.46692	0.1476061

1.2. Preliminary Tests

The results derived from this component constitute the primary focus of the application. It is imperative that the results presented to the user are precise. Given the early stage of the project, akin to several other preliminary tests undertaken for this endeavor, our team opted to conduct tests during the initial semester. These tests involved comparing our results with data provided by our NASA associate and a tin (metal) whisker expert. Additionally, we sought feedback directly from our clients on the results.

5. POV Controller

1.1. Functions and Interfaces Implemented

A simple camera controller that allows the user to move and spectate the regular simulation in real-time.

Quality of life improvements were made to the system such as the speed of the camera that pushes it to be a more reliable tool.

1.2. Preliminary Tests

User testing was used to determine what quality-of-life changes needed to happen to make it useful.

6. Simulation Setup

1.1. Functions and Interfaces Implemented

Simulation setup involves many parts such as starting the whisker simulation subsystem, calling the proper scenes and buttons used for different parts of the simulation, and calling the results handler after deallocating the resources used.

1.2. Preliminary Tests

User testing revealed bugs with a few of the buttons, they could trigger multiple simulations simultaneously which caused rendering issues.

Unit tests provided details on how whisker generation happened, and what scenes were called. The reason for this testing is that unknown scenes were not being deallocated when required and thus the simulation would suffer performance issues.

7. Security Technical Implementation Guides (STIGs)

1.1. Functions and Interfaces Implemented

The repository now contains the checklist and tools to open and edit the checklist. The purpose of the check list is to make sure that the guidelines are followed closely and to give the ISSO an easier way to integrate the tools built for the project into secure areas.

1.2. Preliminary Tests

The process to determine if a STIG was appropriate for the project comes down to this method. We have three choices, N/A, Not a Finding, or Finding. On the basis that our project is a standalone application for desktops/laptops, many issues were N/A such as web-based code or mobile code security testing.

The few cases where it did apply such as adding in ranges for the application we had already resolved thus these are Not a Finding. Lastly, any other cases talking about stuff like security testing like “fuzzing” will be marked out for the ISSO to perform the tests and verify the application in the future.

VII. Description of Final Prototype

Currently, the final prototype is the initial release 0.1, all files are available on Git Hub. This is the process to run the application:

Once in the application (Unity MainApp scene or the executable), click the load button to load in a PCB (printed circuit board). The load button will ask for two files, an OBJ and an MTL file. Make sure these files are the same PCB or issues will occur. The conductive components, position, rotation, and size of the board can be configured in the Board Settings tab. (Test files can be found in the PCB Files folder.)

You can navigate the scene by holding the Right Mouse Button and pressing WASD for direction movements and SPACE and Control for up and down movements respectively.

After the PCB has been loaded in, it will be visible in the simulation. This is where changes to the parameters in the simulation may be made (under Simulation Settings). This includes the spawn area size, the number of whiskers in that spawn area, sigma, and mu for the lengths and widths of the whiskers, among other details.

Once satisfied by the parameters, click the Run Sim button or Run under the Monte Carlo button.

After the simulation has run, the "Preview Results" button will show options to preview different simulation outputs (along with the input parameters for the simulation). The actual results files can be found in the 'SimulationResults' folder.

VIII. Product Delivery Status

The project is available on the Github repository as well as a downloadable executable form: https://github.com/WSUCptSCapstone-S24-F24/-mda-unity3dapp-

Any issues with the executable please inform one of the members of the team.

- Beta Version 0.1
- Beta Version 0.2
- Final Deliverable 0.1

IX. Conclusions and Future Work

Here is a list of future work that any succeeding teams would want to look at to further improve the simulation.

- Save a set of conductive materials for a board, so the components don’t need to be set every load

- Scene UI fixes for organization and workflow
- Use built-in features for the results previewer UI/scroller to improve efficiency with previewing large files
- Create formal unit and system-level tests (and fix resulting bugs)
- Use GPU or other form of parallel processing to speed up the Monte Carlo simulation and allow for more sims
- Optimize viable ranges so that users cannot break the system
- Fix the application side UI so that it scales better and is more visually appealing and user-friendly
- Fix conductive components toggles when leaving and returning to the board settings page

X. Acknowledgments

Thank you to Donna Havrisik who supplied the team with this opportunity and incredible depth into the processes of managing such a project.

Thank you to Jay Brusse, our technical mentor who provided much insight into the issues of metallic whiskers and brought solid directionality to where the project should be focused on.

Thank you to Ananth Jillepalli who ensured progress and held immense patience for the team and its progress.

XI. Glossary

Tin Whiskers: Electrically conductive, crystalline structures of tin that sometimes grow from surfaces where tin (especially electroplated tin) is used as a final finish.

Cadmium Whiskers: Electrically conductive, crystalline structures of cadmium that sometimes grow from surfaces where cadmium is used as a final finish.

Zinc Whiskers: Electrically conductive, crystalline structures of zinc that sometimes grow from surfaces where zinc is used as a final finish.

STEP: This is a file type used in computer-aided design (CAD) and contains three-dimensional model data for a variety of design tasks.

PCB: Printed Circuit Board

Node: Any single 3D structure that is part of the PCB and/or components mounted to it that is electronically conductive and is connected to a particular electrical contact or contacts of the circuit

Node-Pair: A connection between two nodes created by a metal whisker

Unity: A cross-platform game engine developed by Unity Technologies [4]

GUI: Graphical User Interface

XII. References

[1] “NASA Goddard Tin Whisker Homepage,” nepp.nasa.gov. https://nepp.nasa.gov/whisker/ (accessed Jan. 24, 2024)

[2] “Galaxy IV,” *Wikipedia*, Jun. 22, 2021. https://en.wikipedia.org/wiki/Galaxy_IV (accessed Feb. 03, 2024)

[3] “Tin Whisker Spreadsheet,” *web.calce.umd.edu*. https://web.calce.umd.edu/tin-whiskers/spreadsheet/ (accessed Feb. 03, 2024).

[4] U. Technologies, “Unity Manual,” Unity, https://docs.unity3d.com/510/Documentation/Manual/ (accessed Feb. 21, 2024).

[5] D. Pinsky and E. Lambert, “Tin Whisker Risk Mitigation for High-Reliability Systems Integrators and Designers Tin Whisker Risk Mitigation for High-Reliability Systems Integrators and Designers.” Accessed: Feb. 03, 2024. [Online]. Available: https://smtnet.com/library/files/upload/TinWhiskerRiskMitigationForHighReliability.pdf

[6] L. Panashchenko, “2009Panashchenko Log Normal L and T” Accessed: Apr. 22, 2024. [Online]. Available: https://nepp.nasa.gov/whisker/reference/2009Panashchenko_Log_Normal_L_and_T.pdf

XIII. Appendix A - Team Information

Trevor Buchanan - buchanan14.trevor@gmail.com - 253-525-0007

Gavin Mockett, absence.and.eternity@gmail.com. 808-278-3404
Feel free to contact me at this email or by text for anything about the project in the future.

XIV. Appendix B - Example Testing Strategy Reporting

As mentioned, this project does not include unit, integration, or system-level tests. Functional testing was managed through manual assurance checks. Security testing was managed through static code analysis and a manual checklist of STIGs. Please see the project GitHub for the results of both.

XV. Appendix C - Project Management

There are several parts to the project management cycle. Team meetings, Client meetings, and Mentor meetings were all important ways to communicate plans. The cycle of our sprints, branching from the main branch, to the sprint branch, to then the individual branch allowed us to focus on significant progress without causing too many issues. One simple issue about Github with Unity is the organization of the UI elements. Many UI changes will cause issues with merging thus taking a clone of two separate branches and making sure it get manually merged properly was not uncommon.

Team meetings:

These meetings were used to check up on progress with the team, scope out any issues with the project, and confirm the timeline at which changes would be happening. These meetings were performed on Microsoft Teams for ease of access. The team uses the issue board to check what needs to be done and often updates the board during this time.

Client meetings:

This was performed every week with Jay and Donna to get client feedback about the progress that is being made, and figuring out new solutions or requirements as they come. These meetings were conducted over Microsoft Teams again for ease of access, and also for the clients' security.

Mentor meetings:

These meetings with Ananth Jillepalli were also performed every week to discuss the progress of the team and to get a better idea of what the team will need to be prepared for in the future. These meetings were conducted over Zoom.