April 2012

Applying Game UX Techniques to Network Security Tools

Andrew John Lindstrom
Worcester Polytechnic Institute

Eric C. Walston
Worcester Polytechnic Institute

Michael Gheorghe
Worcester Polytechnic Institute

Project Number: MBJ-0004

APPLYING GAME UX TO NETWORK SECURITY SOFTWARE

A Major Qualifying Project Report:

submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

Eric Walston

Mike Gheorghe

Andrew Lindstrom

Date: April 26, 2012

Approved:

Professor Brian Moriarty, Major Advisor

Professor Mark Claypool, Major Advisor

1. Core
2. Network
3. Interface

Abstract

This project describes the design and development of a game-style interface for Core Security's Core Impact Pro software. We identified ways to improve the current interface with commonly used techniques from games and developed a prototype in which we implemented these techniques to make Core Impact Pro easier to use and understand. A user study with the prototype showed that users rated our interface better than the current interface and within twenty minutes of use were on average able to answer more questions regarding the state of the network accurately.

Acknowledgements

We would like to thank Andrew Rappaport our liaison and advisor from Core Securities for his help and guidance during the development of this project, Beth Henkel for her excellent work defining the artistic direction and asset development, Lyndon Goodacre for his role as producer for the project and his contributions throughout its entire duration, Core Securities for the opportunity for this project and their support for its development, and our advisors Mark Claypool and Brian Moriarty for their guidance and direction that helped make this project a success.

Table of Contents

1. Introduction

Core Impact Professional (Core Impact Pro) is a piece of software that is purchased by companies to help ensure that their company's network is secure. The software is limited to operating on a specific IP range that is custom-tailored to the customer's needs. This is the range that the company's security professional hacks into to see where the network's security flaws are. Once the flaws are determined, they can be fixed.

To initialize the process of determining the network's exploits, the software attacks the company's network and attempts to break in, revealing the vulnerabilities of the system. This process is known as "ethical hacking." If the attack is successful, the vulnerabilities are reported to corporate management so that they can be addressed by IT and network security.

Core Impact Pro creates a workspace on the computer it's running on called the localagent. This agent initiates a Rapid Penetration Test (RPT) on its assigned IP range, collecting information about the network. Any servers discovered on the network are reported back to the localagent. This initial scanning phase does not attempt to penetrate the network; it merely enumerates the available hosts and ports, operating systems, IP and MAC addresses and any other data available.

After this information-gathering stage, the user can start the network attack and penetration RPT. This is when Core attempts to hack into host servers and install its own agents, which are essentially Core Impact Pro. By installing an agent into another machine the user is given remote control privileges on the machine which range from normal user to full root access, depending on the vulnerability of the system. A system infected with a Core Impact Pro agent can be severely compromised, allowing the remote user broad access to exposed data and resources.

Once Core has successfully installed an agent on a host, that agent can act as a new localagent, initiating a deeper network scan from the perspective of the infected machine. This ability, known as “pivoting,” is one of Core Impact Pro’s key features. When the Core Impact Pro is finished running and shut down, all modifications made to that system, including the agents that were installed, are removed.

Core Impact Pro’s testing suite is systematic and thorough. It is capable of detecting and attacking any unsecured asset on a network, including PCs, mobile devices, web servers and email systems.

Despite its power and potential effectiveness, the text-heavy user interface of Core Impact Pro suffers from high complexity. According to Andy Rappaport, Architect for Enterprise Products at Core Security, many users of the product are unable to access key features such as the pivoting capability. For this reason, Rappaport approached WPI in summer 2011 with a proposal to make the software more approachable by replacing its text-based display with a game-like user experience.

User interfaces for complex technologies often lack simplicity and intuitiveness. Such interfaces can contain too much information. Large amounts of information at once are detrimental as it clutters the screen with data, buries some data deep into menu systems, and makes important information hard to obtain. To alleviate this problem of excess information in user interfaces, a “Game User Experience” or “Game UX” can be used. A Game UX condenses information into graphics and spatial layouts. The Game UX also uses techniques in gaming to simplify the interaction in an attempt to create an easier, more intuitive experience.

Our design for this project was to take the techniques used in a game UX and apply them to the wealth of information that is displayed by Core Impact Pro. We began by brainstorming possible game related ways to represent the information and workflow of Core Impact Pro. After discussing several possibilities, we believed that it would be better to simply use the display and interaction techniques found in games without trying to enforce a specific goal like a standard game experience. This gave us the freedom to map the information from Core Impact Pro accurately and intuitively without adding unnecessary elements. Because our software does not have a defined goal, it can be better defined as a game-style interface rather than a game. The artistic style was chosen and developed with a similar mindset where the intent is not to deliver a game but an interactive and visual interface for a network security tool. This inspired a simple minimalist representation of the network and its status focused on easy interaction and understanding. We then proceeded to design and build a simple game engine to support the following prototype development. We iterated on the design with significant input from Andy Rappaport and our advisors until it was in a stable working state. We then performed a user study to evaluate the effectiveness of our interface.

The user study had users attempt to answer a set of questions about a simulated network utilizing either our prototype interface or the current interface for Core Impact Pro. Our results were very positive with users rating our interface better than the current interface and within twenty minutes of use were able to answer more questions accurately with our interface than with the current interface for Core Impact Pro.

2. Background Work

Related work falls into several categories. First, there has been at least one other attempt to put a graphical wrapper on automated penetration testing (namely, Metasploit's Armitage extension [1]). Second, a number of videogames with hacking themes have appeared recently, such as *Uplink* (Introversion Software), *Hacker Evolution* (Exosyphen Studios), and *System Protocol One* (While True Fork). Third, network security is often taught via penetration testing competitions that are essentially organized as games. Finally, at least one attempt has been made (in the CounterMeasures project) to develop an interactive, game-like tutorial system for network security training.

The connection between network security and games goes back quite some way and is worked into the grain of penetration testing, as evidenced by the methodology and terminology. In a penetration test, a "red team" and a "blue team" compete to breach or defend a network's integrity, respectively [6,7]. The team names are derived from live military simulations as practiced in the US, with the oppositional "red team" revealing holes in preparation. In deeper history, 19th century German military simulations gave rise to the hobbyist war game community, which later drove the creation of several modern game genres (Dungeons and Dragons-style role-playing, real-time strategy, etc.) devoted to entertainment.

Various entertainment videogames have themed themselves on hacking, and some of these are outlined in the CounterMeasures paper by [2]. The gist of the genre is to use fanciful and often downright unrealistic terminology and game mechanics while giving some sense of the fun and mystique of attacking or defending a network. Typical mechanics are building up funds from successes to buy new "viruses, " or alternatively, "memory" and other upgrades that will defend against viruses. Usually the games progress via missions or levels, and sometimes they include distinct tutorials. The total amount of useful knowledge that will be gained from playing these games, as applicable to network security, is generally quite low, though they may be very entertaining to play.

Penetration testing is a serious business. Mistakes can bring down production servers and even destroy databases (not to mention, on the other side, overlooking vulnerabilities and allowing the theft of critical information). This is one of the reasons that the testing usually takes place inside a given time window when the system administrators (the blue team) are aware that the machines in their network could be under attack.

A professional red team inflicting collateral damage is likely to lose business. Thus, more than in other domains, it may be useful to carry out training inside what in game studies is referred to as a "magic circle," an area dedicated to experimentation and exploration without consequences that pose major threats. Network security is often taught with the aid of penetration testing competitions. In these large-scale games, any damage caused is unlikely to reflect on company performance and revenue. Significant competitions take place in all areas of the country.

Among competitions, the National Collegiate Cyber Defense Competition (NCCDC), operating since 2005 and recognized by Congress for promoting cyber security education, is the nation's largest [5]. Eight yearly regional competitions serve as qualifiers for the main event. The Midwest Collegiate Cyber Defense Competition (MWCCDC), itself sponsored by the Department of Homeland Security and the US Army, is a virtual qualifying competition for the NCCDC [8]. As such it represents both a serious training ground, one from which curricula can be gauged and improved and students can be hired by corporations, and an innocuous mock-up. Students are responsible for defending their virtual network from attack by outside hackers.

The CSAW (Cyber Security Awareness Week: pronounced "see-saw") competition is a yearly event hosted by NYU-Poly and open to high school students and university students from all around the world. Over the span of two days, competitors respond to a variety of challenges. One major component is a Capture the Flag game emphasizing hard security skills as well as strategic team-building. This derives from the original and now infamous DEFCON Capture the Flag security tournament. [3]

Although these long-term projects involve training, games, and professional network security, and perhaps come closest among all cyber security competitions to our project, they are otherwise quite different in motivation. Specifically, we aimed to build an interface that makes network security teaching concurrent with network security testing. The target audience is IT professionals, but not necessarily professionals well-versed in security.

The current version of Core Impact Pro (version 12) includes extensive information on the testing modules that can be run. Documentation is complete and includes functional ToolTips. For experts, the source code for the modules can even be examined and altered while the software is running. Core Insight, a related application from the same company, uses sophisticated artificial intelligence routines to automate the entire process for users who are less familiar with the technical details of security. Insight represents a parallel effort to improve ease of use, but one less concerned with revealing the details to users over time.

More directly aligned with the present effort is Armitage, the graphical wrapper for Metasploit, an open-source penetration testing tool. The philosophy and goals of this project have some similarity to ours, but with little emphasis on a game-like experience in particular. Armitage does include a "multiplayer" option to chat with teammates and share a workspace remotely. (It is even possible to share the same remote shell in multiplayer mode.) Networks are shown graphically to supplement the information made available through Metasploit. The target user is still assumed to have domain expertise, however, and the bulk of the information is conveyed with text. [1]

A tool similar to both Metasploit/Armitage and Core's offerings is BackTrack Linux. Like Core's software, it includes and expands upon the Metasploit tools. Though most of its tools do not use graphics, its menu-driven GUI is intended to improve ease of use. Several tools (for example, *zenmap* and *wireshark*) do have significant graphical components. [4]

On the side of training rather than professional application or usability per se, a previous Major Qualifying Project at WPI, CounterMeasures, looked at the possibility of using videogames as training devices for network security techniques. Tutorial missions were developed around a sandbox shell environment. Players received clear goals and were allowed to learn by trial and error. It was hypothesized that, tested in a real shell environment, they would be able to accomplish the same steps in less time after training in the videogame as compared to training with a textbook. A small user study supported this conclusion.

3. Development

The development of our entire project began with meeting Andy Rappaport from Core Securities and understanding our overall goal. We then discussed the design and laid out our approach for developing the new interface. This began with researching and developing a simple game engine utilizing some open source libraries. Once we had the engine built to a usable state we began developing the interface prototype. We continuously refined our prototype for the remainder of the project with suggestions and feedback from Andy Rappaport and our advisors. We concluded with a user study to judge our prototype's effectiveness as an interface for Core Impact Pro.

3. 1 Initial Project Approach

One of the first meetings as an MQP group was to discuss and understand the needs of Core Securities and to design a game that would meet those needs. This became a collaborative design discussion over several meetings where we were able to decide some of the major details of the project before we began development. This step was crucial to the success of the project as it provided the necessary foundation for the development of the interface.

One of the first decisions we came to was to not make the project into an actual game. A game needs an achievable goal and Core Impact Pro does not exactly fit that formula. A standard user of Core Impact Pro will attempt to hack into the computers across a network with the intent of finding any security holes and how those holes were breached. While the user does have a goal in this case it is not well defined and changes as the user discovers vulnerabilities. Ideally the user would attempt many exploits and be unsuccessful at installing agents on all of the machines on the network. This can be considered a very successful and hopefully common occurrence on strong and secure networks. To attempt every permutation of every exploit would take far too much time so a goal of exhausting all possible actions is not feasible. Instead, just ensuring that many exploits aimed at a system were unsuccessful is a good "win condition" but it is up to the user where to stop. If a user were to discover vulnerabilities on the network then they may want to simply stop there and fix the issue or they may want to see all the vulnerabilities that they can exploit. Some users may just want to see if certain machines can be infected and some may only be interested in the machines on a specific subnet and not be interested in exploring all the available machines. Because of this extreme variety of use cases, we decided that it would not be beneficial to a user to give them a specific goal to achieve. Instead we decided that the focus of this project would be on developing a game-style interface for the interactions available in Core Impact Pro.

The difference between a game and a game-style interface is that the interface will provide a user experience similar to a game but not the goals that are associated with a game. The game user experience or Game UX is ideal for Core Impact Pro because it will help filter the wealth of information while focusing on visual feedback and direct interaction to hopefully make working with the interface easier and more intuitive. Core Impact Pro’s interface contains a large amount of information that is all displayed to the user through lists and tree menus which can be useful to an experienced user who knows exactly where to look for the information they seek but this can seem cluttered to new or inexperienced users. A game interface would display the more basic and necessary information, such as how many machines are on a network, but will only show more detailed information when the user interacts with the different network elements. Visual feedback will also help the user have a better understanding of the context of their actions. Allowing the user to initialize modules to run on the network by directly interacting with the visual representations of those machines on the network will also help improve the users understanding of context. These were the key aspects of a game interface that we wanted to apply to Core Impact Pro and making this distinction early on helped guide development away from a game and towards improved understanding though game-style interactions.

Our team consisted of three undergraduate Computer Science and Interactive Media and Game Development students who would be in charge of developing the prototype and one graduate Interactive Media and Game Development student who would coordinate the production of the prototype. This team composition was heavily programming oriented and our design needed to take that into consideration. This meant we would not be able to develop a prototype that required a significant amount of artistic assets or animation. This led to the decision to develop a two dimensional game with a simple interface and a game map similar to common real-time strategy games. This style would display a main map with directly intractable game elements framed by more detailed information and options for the user in menus. This decision was coupled with the idea of using a minimalistic art style with a neon wireframe look. We believed this decision would make art asset development easier as well as convey a good middle ground between a playful videogame and serious network security technology.

Our inspiration for the gameplay and interface came from tower defense and real-time strategy videogames. We liked the way these interfaces would typically show an overview of the battlefield with many graphical indicators of state such as health bars. These interfaces also would have menus that frame the play area with more detailed information and options. Unfortunately these types of games often have a fast-paced aspect and the interactions would need to be tweaked to fit the flow of Core Impact Pro. When using Core, a user will begin modules and as those modules find information the new data is fed back into the interface so the user can get a better understanding of the network and be more informed for running more modules. Our game interface would need to reflect this flow of actions, followed by a period of processing, and then asynchronous results. We discussed the possibility of an adversarial approach where the absence of information was framed as an enemy that would be destroyed as more information was gained but this approach also would not serve the flow of Core Impact Pro. If we consider a successful case where a user is unable to gather much information about a network, the idea of success being tied to the strong existence of an enemy did not make sense. Similarly when discussing the possibility of the user playing as the "bad guy" trying to break into the network, we felt this would put the user in the wrong mindset. In the end we chose a less game-like approach where we attempted to portray the state of the network accurately and simply. We would show towers as entities on a network, connections between these towers representing a connection on a subnet, and attacks as a pulse along those connections. There would not be an adversary, the available information about each network entity would appear as the updates were received from the modules, and actions such as scans or attacks would be represented by repeating animations until the modules that they represent had completed.

We made sure to make these design decisions early to guide our development in later stages. These aspects of the design were key in deciding what game development libraries to utilize and what considerations to make when developing the engine and initial game interface. This was the spearhead of our approach to completing this project and laid the groundwork for our development.

3. 2 Selecting and Developing the Technology Stack

Once we completed the initial design of the game and were set on some of the major details it was time to look into which technology we should use to build the game. We decided that this would be a two dimensional game relatively simple graphics. The focus of our development would be on keeping the information manageable and providing an interactive interface that was intuitive to users. We also knew we would need to be communicating with the Core Impact Professional python library asynchronously so it would be beneficial to choose an engine that would make this easier to deal with.

We began research on game engines and technologies that would help us develop the prototype. After looking into several engines that specialized in supporting two dimensional graphics development and even considering attempting a three dimensional prototype we settled on five development platforms that we could choose from. These were SDL, OpenGL, SFML, C4, and Allegro. We wanted to utilize the strengths of our team and keep ourselves in familiar territory to hopefully make developing as smooth as possible. We felt that developing in C++ would be best due to our familiarity with the language and its inherent speed. Each of these engines had pros and cons for our development but in the end we chose to utilize SFML for handling our graphical interface. From there we would supplement SFML with other libraries and our own code to have just the functionality we needed.

Our final decision was to build our own simple game engine for scene, asset, and game object management on top of the Boost C++ Libraries, the Simple and Fast Multimedia Library (SFML), and a GUI system aimed at game interfaces called GWEN that utilizes SFML's rendering component to render GUI elements. Together these libraries form the base of our technology stack that allowed us to develop our engine quickly and begin development of the prototype early on. Boost provided many useful general purpose libraries as well as providing a way to communicate with Core Impact Pro's Python library. SFML provided asset loading and rendering as well as viewports and window functionality. GWEN was added to simplify the development of the menus and interface elements that were not sprite based.

This technology stack made the development of the prototype easier by providing tools to simplify some of the more time consuming elements while still allowing us to have complete control over the design of the game engine and its functionality.

3. 3 Stages of Development

Once we understood the initial design of the interface and what technology we would use to develop the prototype we were able to begin the stages of development. One of the first steps to beginning development was setting up a version control system for maintaining our code and assets. We chose to use SVN for familiarity and our access to Teamforge through WPI. This would be our primary storage location for all assets, code, and anything else related to the project that we wanted readily available to its members. The next step was to take the libraries that we chose to use for development and make a project with those included. This involved compiling the Boost, SFML, and GWEN Libraries and linking the all to a basic starting project. We also set up a simple Google Sites webpage to track our progress and maintain a prioritized task list to organize the actions of the development team. Over time this website was used less and less but it helped organize the early stages of the project.

The project went through six stages throughout the four terms of development ranging from the initial design of the engine and the prototype all the way through the user study to evaluate effectiveness. Each of these stages marked an important change of focus for the team and brought the project closer to completion.

3. 3. 1 Engine and Prototype Design

The first stage of our development was working out the design of our interfaces interactions and artistic style as well as the design of the engine that would support this design. The engine was designed and developed primarily in A-Term while the prototype design came towards the end of A-Term with the artistic aspects occurring at the beginning of B-Term.

3. 3. 1. 1 Engine Design

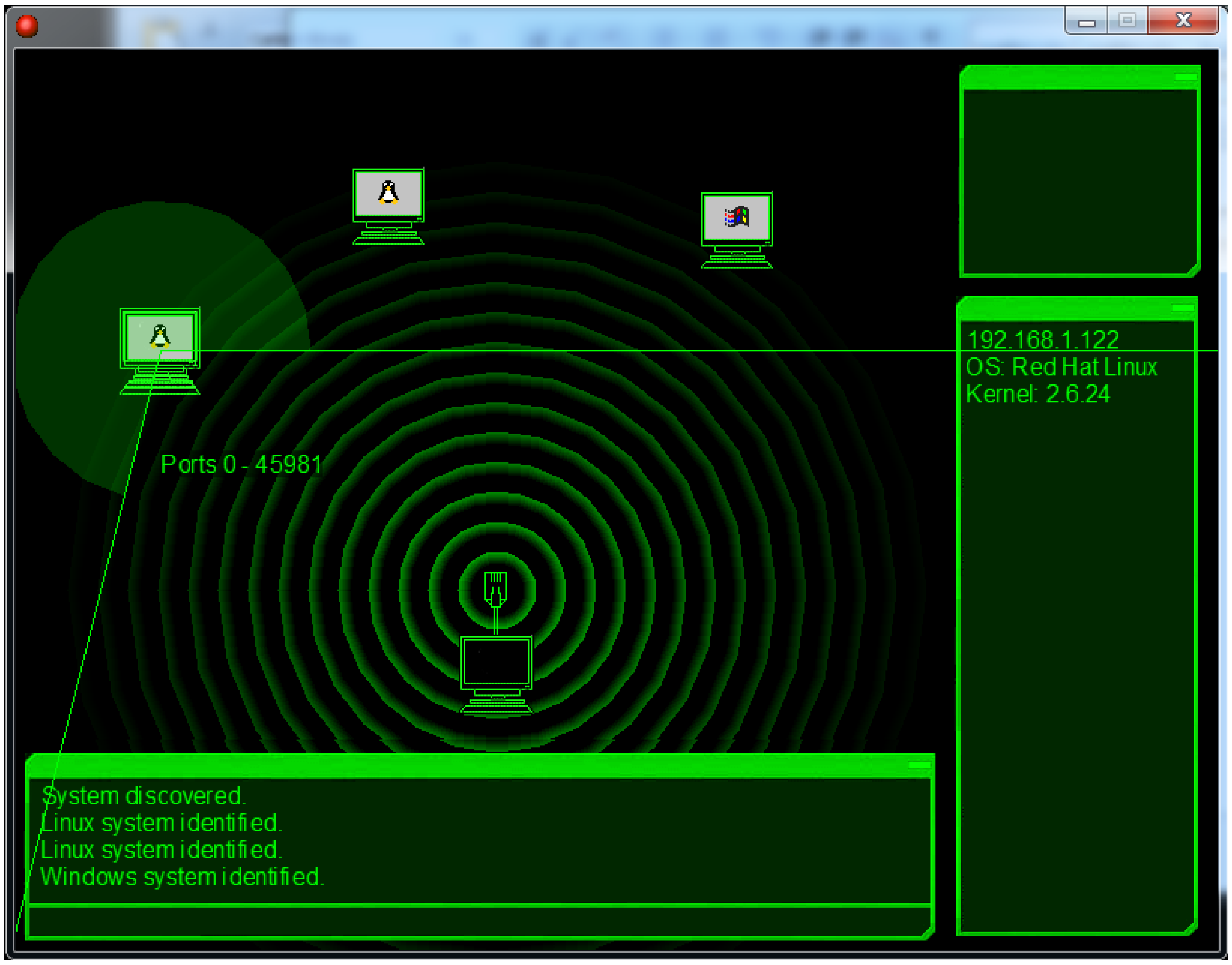
The design for the two dimensional game engine was one of the first development oriented actions we did as a team. We understood the basic elements we would need to handle such as sprite and sound management and the bigger aspects of game object management and draw order. We began with a simple UML diagram to coordinate our design and focused on the most important aspects that we would need to make the interface. While this diagram was not kept up to date for the current version it does represent our early thought process of how the game engine should function which is in the appendix. Our design was set up to handle game objects, images, sounds, animations, a simple scene graph for draw order, and a way to represent the initial state of the current project.

The developer would create an instance of an Application that would be loaded in at the beginning and defined the game objects that initialized the game. Each game object would be kept in the current Level which stored all objects in an ordered array by z-value which would determine which order they are drawn in. Each game object would have its update and draw methods called for each frame. If a game object would play a sound or draw a sprite it would initialize the asset by calling the proper function in the singleton AssetManager which would maintain a list of currently loaded assets and ensured that game objects that shared assets would reference the same loaded data. These simple aspects were the foundation of our game engine. There were many small tweaks to the functionality of the engine over the course of the project but these aspects were the core components that made the development of the game interface faster and easier.

3. 3. 1. 2 Preliminary Prototype

Once we had developed a clear vision for the elements we wanted in the interface it was time to create a preliminary prototype that reflected those elements. The purpose of this prototype was to make sure Core Securities, the development team, and our advisors were all on in agreement with the look and feel of how the interface would work. This was a very important step early on because it helped us identify the different aspects of the interface that we would need to program. The discussions of the preliminary prototype helped us get a sense of the amount of information we would need to handle, where it should appear, and how users would be able to interact with it.

We developed our preliminary prototype in Game Maker which is a simple game development tool that is very useful for making quick and easy prototypes of two dimensional games. We made an example scenario where there was a machine on a network and the user could perform a scan to find other machines on the network (Figure 1). When a scan was initiated a pulsing animation began and after a short period of time other machines appeared representing other computers on the network. As the scan continued symbols representing the operating systems of the different computers appeared. The user could then click to initiate an attack on one of the machines which showed a red ball shooting at the target machine. The interface also had feedback about what actions were being performed and what information as available.



System discovered.
Linux system identified.
Linux system identified.
Windows system identified.

Figure 1 – Game Maker prototype interface

This prototype helped our entire group discuss the look and feel of the final interface and guided our development of the project. We liked the menu layout with the feedback information as well as the direct interaction with towers and representative animations. These elements became major goals of development and can be seen in the final product.

3. 3. 1. 3 Artistic Design

The design for the look and feel of the prototype went through several iterations in conjunction with our artist Beth Hankel during the second term of the project. The art assets shown in the Game Maker prototype were only temporary and we looked into how to design the proper theme to give the user an interesting game-like experience while still conveying the serious nature of the interactions and the product they were using.

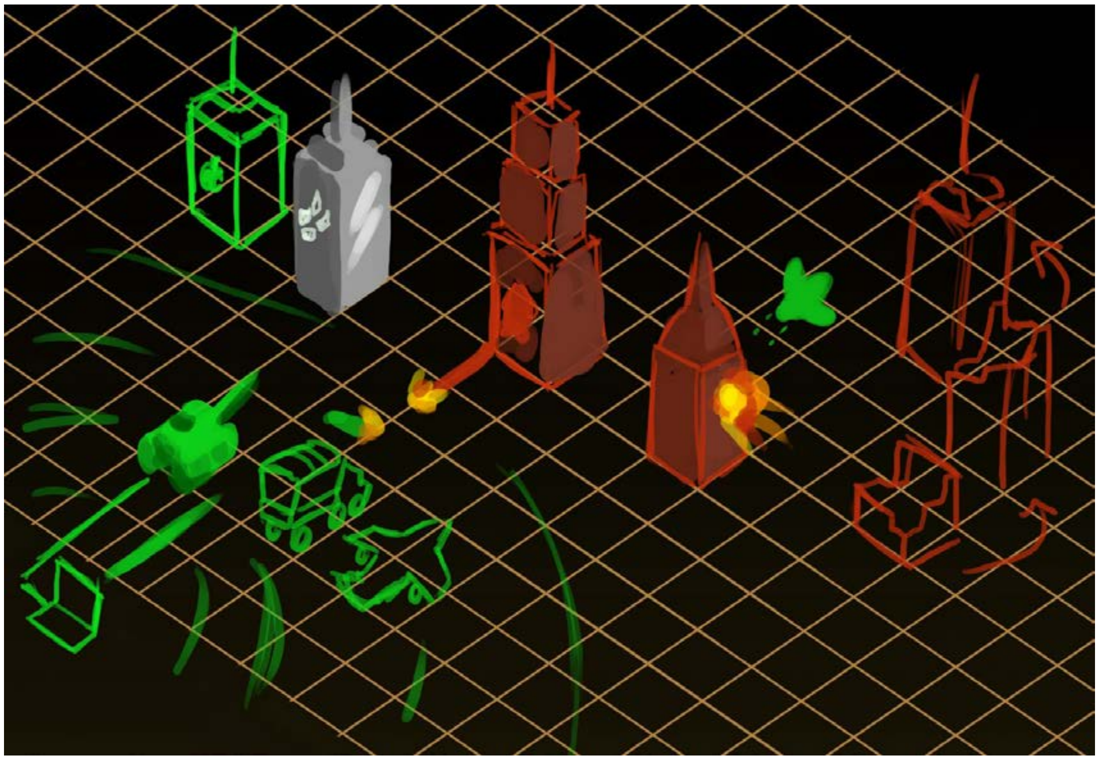


Figure 2 – Early war-themed concept art

The concept art shown in Figure 2 was an early design for the look of the game and shows buildings representing machines on a network being attacked by military style vehicles originating from a computer. We liked the use of buildings representing machines but the use of military vehicles made it feel too much like a game that would not be taken seriously. It was very important for the nature of the software to reflect the fact that the users would be interacting with a network weapon. In future designs we got rid of the military style but kept the tower aspect of the design. The grid and isometric view also seemed to work nice for displaying a view of the network with a large amount of information.

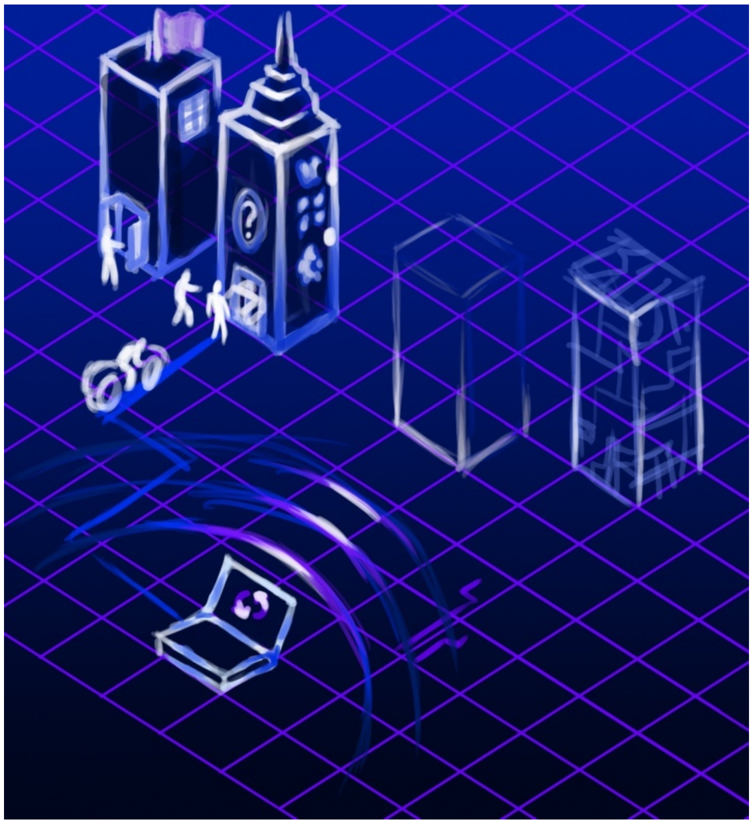


Figure 3 – Early espionage-themed concept art

The next iteration of concept art shown in Figure 3 had more of an espionage theme where we see agents sneaking into buildings and gathering information as well as taking over the machines. This design kept the isometric view, grid component, and towers representing machines on the network. It was an improvement but even the espionage feel seemed too game-like to portray the serious feel of penetration testing.

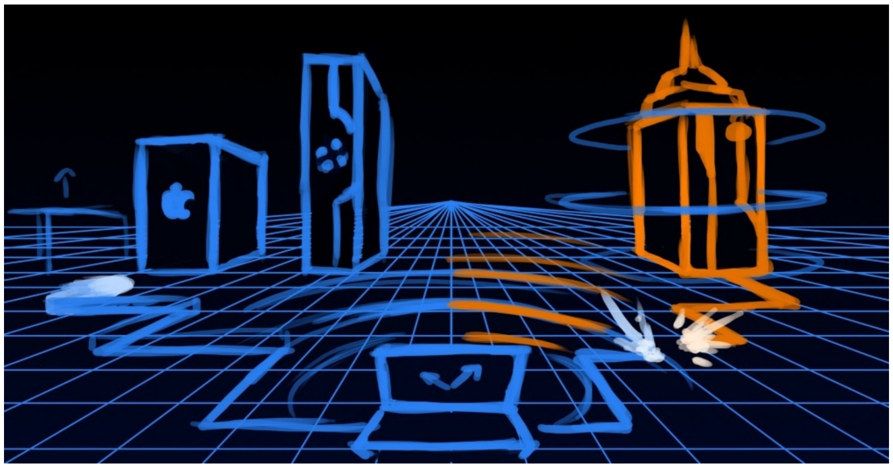


Figure 4 – Concept art with third person viewpoint

This next design shown in Figure 4 explored a different camera angle and removed the idea of an actual attacking agent entirely. This portrayed the serious aspect that we were looking for and we liked the way information was able to be portrayed but decided to return to the isometric view while keeping everything else from this example.

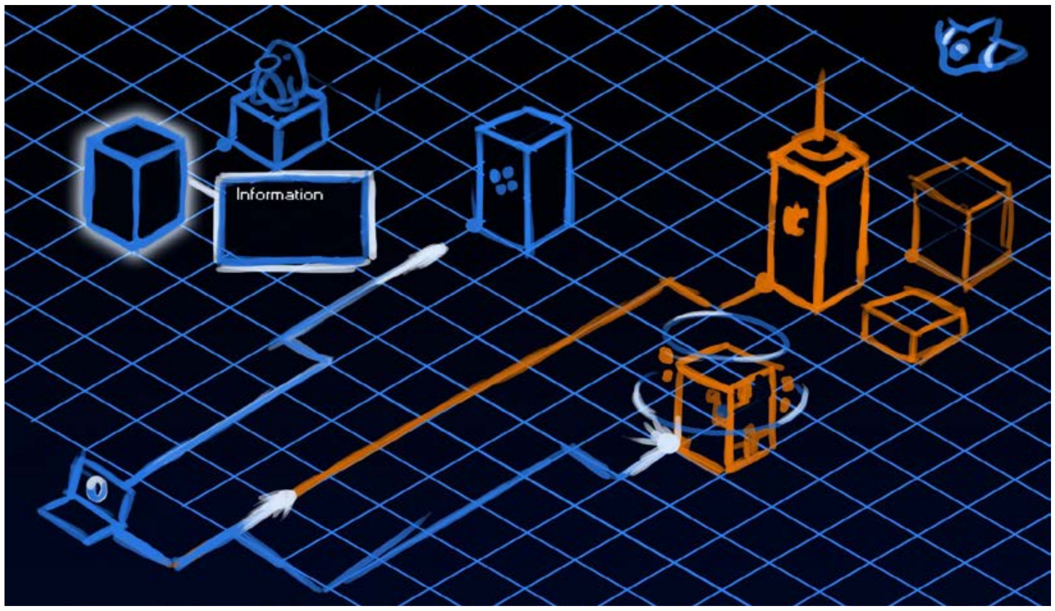


Figure 5 – Final concept art

The final concept art shown in Figure 5 became the basis for the artistic design of the project. Towers with identifying features represent machines on a network and lines on the grid represent network connections. Boxes appear hovering over the grid with detailed information and pulsing animations represent different actions being performed on a zoomed out isometric view for displaying a decent sized area for representing the network. These are all aspects that made it into the final game as well as a similar coloring of the environment. This concept art was a key component for developing the artistic design of the project.

3. 3. 2 Engine Development

We decided through preliminary research to use some free open-source libraries to supplement the engine and make development faster and easier. We used the Boost libraries to keep track of the game time, to communicate through python with Core's library, and for some other general purpose functionality. The Simple Fast Multimedia Library (SFML) was used for most of the asset management and rendering. SFML provides an interface for loading images and sounds that can be rendered and played through the provided window. It also provided a user input interface, draw-able strings, and a simple way to zoom and move the screen which we took full advantage of. SFML also provided the render target that is used by the GWEN library. GWEN (not an acronym) is the third library that we used to create the user interface elements such as text boxes, buttons, tree menus and the different menus within our interface. All of these libraries kept our interface independent of the operating system which was not required as part of this project but is certainly ideal for potential portability.

In addition to the parts of the engine already discussed in the design section we added two major elements over the course of our development. The first was with the asynchronous communication with Core's python library. This could be considered part of the prototype rather than part of the engine but this required buffers of messages that were communicated through Boost's Python interface and only the received message were used as part of the prototype. This aspect allowed us to send XML modules to Core Impact Pro and for us to receive XML responses indicating additions to the current state of the stored network information.

Another major aspect was an XML interface initializer that was used to define a layout for the GWEN interface elements. This read in an XML file and built the interface out of GWEN elements that were represented by specific tags and initialized with the given attributes. This made it quick and easy to define and edit the interface elements in the prototype. This speed was very useful during the tweaking phase where we spent time making sure the interface felt polished.

3. 3. 3 Prototype Development

The development of the interface prototype was by far the longest stage of development lasting from the end of A-Term through to the beginning of D-Term. This encompasses all of the code development that was specific to the implementation of the final product. This was the focus of the team from when we were able to compile the almost completed engine with the supporting libraries until all major features were implemented.

Development began with setting up a simple interface with a few GWEN windows for displaying information and a main screen with a grid to represent the current state of the network. From here we wanted to make an image of a tower appear on the grid to represent the local host. From this point we developed the ability to add more machines to the grid to represent other machines on the network as shown in Figure 6.

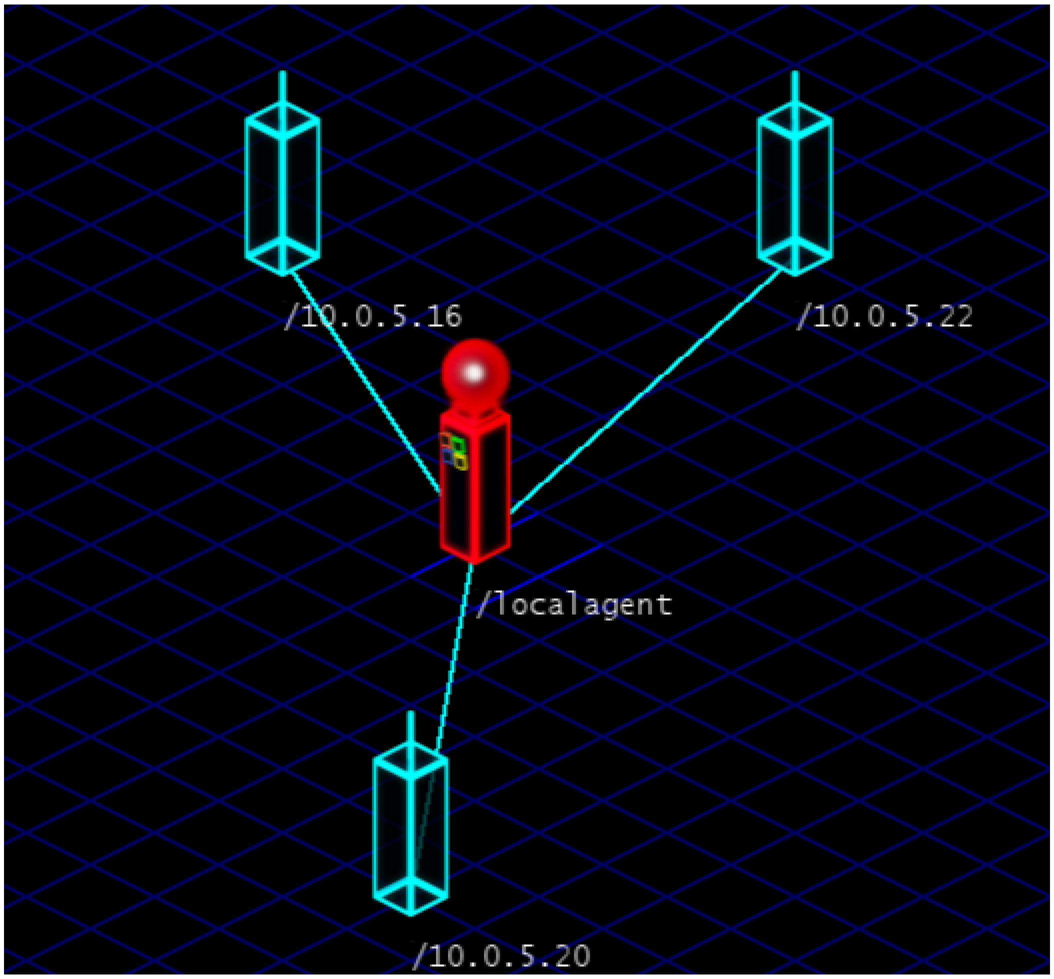
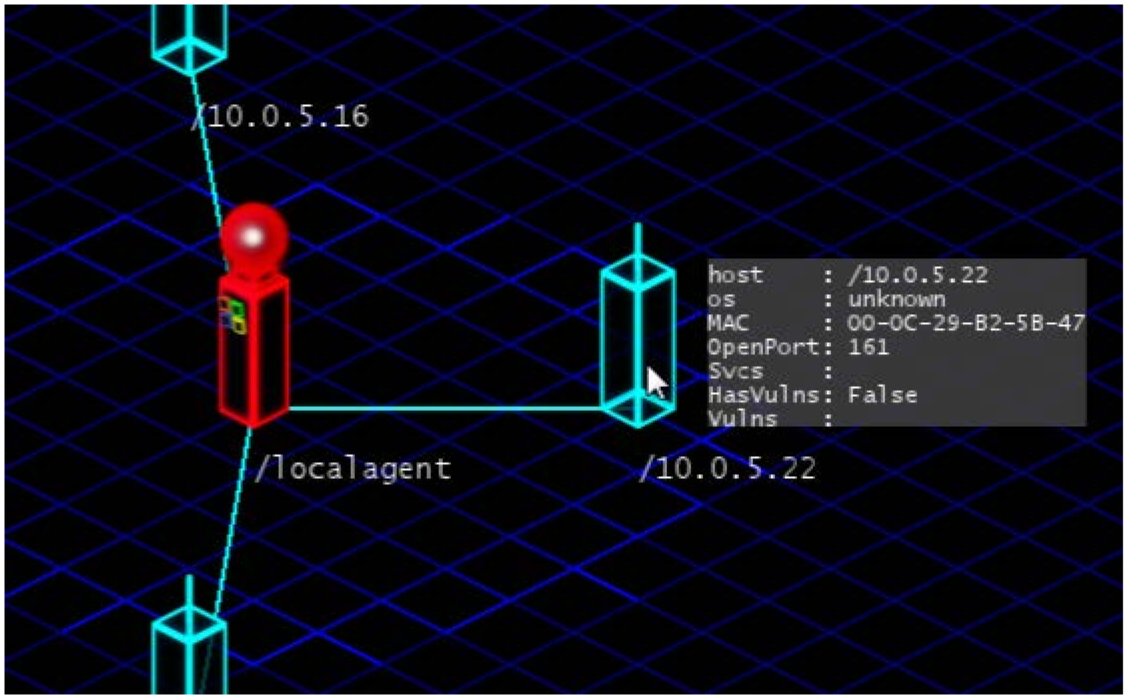


Figure 6 – Towers on a network

Early on it was clear that we would need to develop a system for faking the interactions between our interface and Core Impact Pro. To run Core Impact Pro we needed to be set up with a proper license for the software, get the software installed properly, and set up a network of virtual machines to attack. Even when this was accomplished it was still significantly slower to complete scans and attacks because they had to be run in real time. To solve this issue we developed a ‘stub interface’ along with Andrew Rappaport that would receive the commands we would normally send to Core Impact Pro and send back sample strings that represented what would have been returned by a real call to Core. This early development helped speed up code production and made it easier to decide the format of the messages that would be sent back and forth. This stub interface is discussed in greater detail in Section 3. 4.

Development continued with the implementation of the basic animations for scan and attack, the connections between towers represented by lines on the grid, and the radial interaction menu for running modules from towers. The scan animation was represented by a pulsing wave of lighter blue color on the grid originating from the tower that initiated the scan. This pulsing animation repeats until the message from Core says that the scan completed. The connectivity of towers on the network is represented by bright lines between towers. The animation that represents an attack against another tower is a pulse that travels from the source tower to the target tower along the connection line and repeats until Core Impact Pro indicates that the attack has been completed. The radial interactions menu appeared when a user right clicks on a tower and displays the different modules that can be run. We also represented the operating system of a machine with a badge directly on the tower so a user could easily identify the operating system of a machine at a glance. Similarly, another badge along with a color change to yellow was used to show that an agent had been installed on a successfully attacked machine. These additions were primarily aesthetic but were an important aspect that took the information that was normally displayed as text and represented it visually with direct interactions between the user and the representation.

As we began getting more information back from the modules we started displaying this information in the form of hover displays (Figure 7) and menus in the GWEN windows.



```
host     : /10.0.5.22
os       : unknown
MAC      : 00-0C-29-B2-5B-47
OpenPort: 161
Svcs     :
HasVulns: False
Vulns    :
```

Figure 7 – Hover display

This information can be useful to the user but is not necessary to know right away or see at all times. The hover information was used for some basic detailed information about a machine and when the user selected a tower the GWEN window was used to display all available information about that machine.

The final major additions to the interface came with the implementation of radial menu functions for harvesting information from a tower and pivoting to a target tower and continuing attacks from the new source tower.

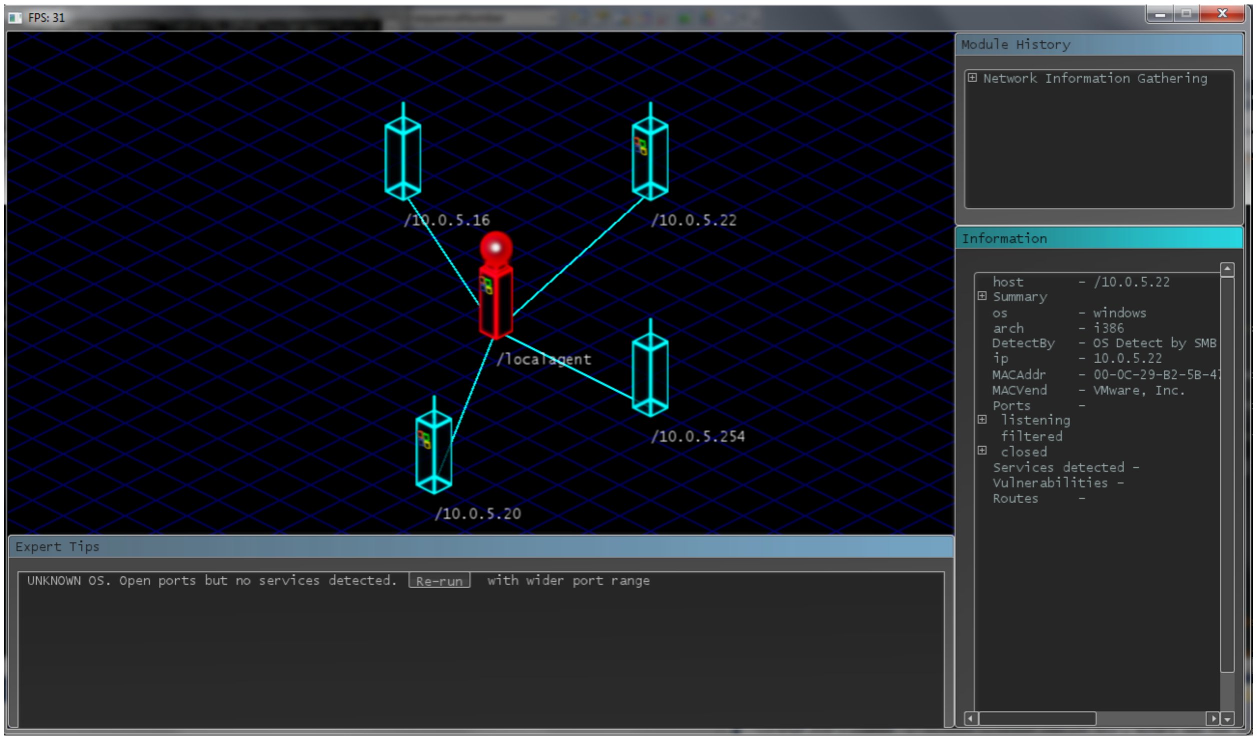
3. 3. 4 Refinements, Tweaks, and UX Improvements

As the majority of the desired features were implemented we began to shift our focus from implementing new features to polishing the interactions that comprised the currently available features. This was a part of the prototype development that occurred throughout C-Term and D-Term and helped improve the way users were able to interact with the interface.

Throughout development we encountered several issues with lag that needed to be resolved before we could finalize our product. These occurred primarily with the interactions between Core Impact Pro and our interface. We found that at times, especially when dealing with the stub code, that a very large amount of data would be received and this would cause our interface to lag significantly while it parsed the data. For some interactions the stub code was accidentally sending about thirty minutes worth of responses from Core Impact Pro all at once which explained the extreme amount of information. Similarly with the live code, data was cached when it was received and our interface would use the Python code to poll for more data. The polling was too infrequent and sometimes this would cause a large amount of data to be received at once. We increased the polling rate and the lag was not an issue.

Initially the radial menu was the same for each tower so a change was made to disable actions that could not be performed from the selected tower and to display a different color menu for a non-active tower. This was a simple change that made sure that an attack was not initiated on the same tower as the source or other similar interactions that were possible commands but would confuse Core Impact Pro and return unexpected results.

The information that was displayed in the module list and the detailed information menu grew to be more than could be easily displayed in the available space so to handle this, the information was stored in a tree control menu which can be seen on the right of Figure 8.



```
UNKNOWN OS. Open ports but no services detected. Re-run  with wider port range
```

```
 host          - /10.0.5.22
⊞ Summary
 os            - windows
 arch          - i386
 DetectBy      - OS Detect by SMB
 ip            - 10.0.5.22
 MACAddr       - 00-0C-29-B2-5B-4
 MACVend       - VMware, Inc.
 Ports         -
⊞  listening
   filtered
⊞  closed
 Services detected -
 Vulnerabilities -
 Routes        -
```

Figure 8 – Reverse L and tree control menus

This let information such as open ports to be stored as a sub tree of a single statement which helped condense the amount of information that needed to be displayed.

Another key addition that allowed us to expand the network graph beyond only a couple nodes was the addition of zooming, and moving the grid. This allowed us to expand the display indefinitely and the user would be able to zoom in and out to see whatever portion of the network they wanted to focus on. The network graph was initially a static representation where each tower would be placed in a specific position. We later changed this to a dynamic layout that would expand radially and distribute the towers evenly so we could have a representation that could handle any number of towers. Along with this addition we also added the ability to move towers around manually and lock them in place which would allow a user to make the graph look like whatever they envision their graph to be.

These additions were not necessary to make an interface that could perform the required functions but they were additions that made the displayed data easier to interact with and more intuitive to read and understand from the users point of view.

3. 3. 5 Guidance Development

One of the final additions to the interface was the development of the guidance text to provide helpful hints to users about what they might be able to do to be more effective when using the product. This guidance text became one of the last major goals for the prototype once all the major functionality was implemented and showed a way that our interface could effectively innovate beyond the current Core Impact Pro interface. The standard interface does not have this guidance text and our interface became a medium for its development.

A system was developed by Andrew Rappaport within the Core Impact Pro python library to look at the information that was being sent back from the completed modules and attempt to provide advice on what could have been done to receive better results from that particular module. This advice was a set of rules intended to emulate the knowledge of someone significantly experienced in penetration testing. The system might see that a scan that was run on a narrow port range was unable to identify the operating system of a machine on a network and could suggest that running a new scan with a wider port range might succeed in identifying that machines operating system. This type of information that an experienced user would know but an inexperienced user might not is an invaluable aspect to both the interface we developed and to Core Impact Pro in general because it can help educate new users and make them more effective penetration testers.

The guidance information was sent back as an XML string to our interface. It contained a string of text to offer suggestions. Additionally parts of the text might be in a special tag to indicate that they should be rendered as interactive. The interactive text would show up as a button and when the button was clicked it would initiate a new module that was also sent as part of the XML string. The guidance text was rendered in the bottom section of the interface as part of the first tier of information that would be shown to the users at all times to hopefully keep them very aware of the suggestions. During the user study, almost all users of our interface utilized the guidance text button to initiate a re-run of an unsuccessful module. This was a very useful aspect of our interface that went above and beyond the current functionality of Core Impact Pro.

3. 3. 6 User Study

The final aspect of our project was to perform a simple study to attempt to gauge the effectiveness of our interface when compared to the previous interface of Core Impact Pro. We decided to run a relatively informal study that would have users use one of the two interfaces to perform a set of common tasks on a network and answer questions about the information gained from those tasks. We chose a list of questions and had twenty eight people run the test split between the two interfaces. We also gathered subjective feedback from the testers about their opinions of the interface. The details of the user study are discussed later in Section 4 of this paper. This test of effectiveness concluded the development stages of our project.

3. 4 Simulated Scenarios and Stub Code

The development process required continuous testing, which proved to be a major hurdle for the project. Testing against Core Impact Pro is a lengthy process due to having to wait for the application to load, and then to create a blank workspace. It also requires our development machines to have a copy of Core Impact Pro installed. In order to shorten the development cycle, we decided to split the necessary testing into two different levels, stub version, and live version. In stub version, we created a duplicate version of Core Impact Pro's python interface that didn't actually communicate with Core Impact Pro, but just maintained a simple state machine and returned placeholder values that were taken directly from the live version of the interface. The data we were getting back was static values, but syntactically and logically correct values captured from the live version. This allowed us to perform most of our testing using the stub version, saving us considerable time and effort compared to working with the live version of Core Impact Pro. It also meant that we did not have to install Core Impact Pro on each of our development machines, and could continue development on any machine that had Visual Studio and TortoiseSVN installed. Getting a perfectly functional stub version of the python interface presented us with additional difficulties.

One of the problems we encountered was that the stub version did not always return all of the XML we were expecting, or had different tags or attributes named differently than what we were looking for in our prototype. These issues were difficult to locate because the Visual Studio debugger cannot debug the python interface, so any time we ran into an issue located within the python interface, we had to capture the input we were sending it, stop, load up the Eclipse Python IDE, and try to recreate the issue with the given input. In the end, those types of problems did not occur often and we still ended up saving a lot of time during the testing process.

The other major issue was that Core Impact Pro cannot be easily modeled as a state machine due to the fact that feedback returns asynchronously from running modules. As a result, we initially captured all of the output from a module and returned the XML all at once. The resulting XML was hundreds of thousands of characters, and was significantly slowing down our parser. We ultimately decided to manually prune the XML so that we only get XML that we specifically want to respond to, which resolved our performance issues.

Testing our code on the live version was also difficult for its own reasons. The primary issue was that Core Impact Pro is inherently a very dangerous program to run on a network. It can cause machines to crash and bring down vital services. And, in order to get any useful results while testing, we had to be testing against a set of machines that we knew would be safe targets. Our solution was to use virtual networks created in VMware Workstation, on their own virtual networks. Our testing copy of Core Impact Pro was locked to only operating on the IP range 10. 0. 5. * and 10. 0. 6. *, so we created two virtual LANs, ran different virtual machine images on each, and set the network adapters to those specific IP ranges. These precautions ensured that we could not do any damage while testing our prototype.

Our final setup had two testing modes, stub, and live. Control over whether or not we were using the stub version was controlled in a configuration file. The stub version contained a state machine that kept track of the XML it received and modified its state accordingly. Every 25 milliseconds we would poll the python interface to see if there have been any new events since the last time we checked, and if there were we would parse them. We found that having the delay be lower than 25 milliseconds cause frame rate drops in our prototype, while setting the delay too high ended up stalling the Core Impact Pro driver, as the entire python interpreter went to sleep and Core Impact Pro paused with it, which resulted in modules taking nearly an order of magnitude longer to complete.

3. 5 The Structure of the Completed Code

Our Mayhem Game Engine uses an object-oriented approach to managing the game. The engine has a GameManager singleton, which maintains all of the GameLevel objects, updates all the GameObjects, and performs all of the necessary initialization and window management that SFML requires. It handles the game loop, measures the frame rate, and allows the user to create new GameLevel objects.

The DisplayManager singleton maintains the SFML window, and provides all the GameObjects access to the window and the current video mode.

The AssetManager singleton allows the user to load new sounds or sprites, and get a reference to the resulting Asset object. The AssetManager maintains a list of sounds and sprites that have been loaded before, and if they have the asset manager returns the existing reference. This is a basic form of resource management that prevents the system from using up more memory than is necessary.

The GUIManager singleton maintains the base GWEN canvas elements and provides a blank slate for the user to add to using GWEN objects. It performs the necessary skin initialization for GWEN, as well as loading the GUI layout from an XML file. The XML file provides the user a way to create elements, place and size them, and add default content to them without having to modify any engine or game code. The GUIManager provides access to the dynamic elements that it creates by storing them in a map, and requiring that each element have a unique string identifier. The programmers can then use this identifier to access that specific element, and check status or register callback functions as necessary.

The TimeManager singleton maintains all of the existing timers. It updates them as necessary, and when the timer expires, the timer object calls the callback function in its associated GameObject.

The next layer beneath the singletons is the GameLevel object. The GameLevel objects store all the GameObjects, and contain all the functions that update the state of the game. The GameLevel maintains the list of objects, and provides functionality to search for GameObjects by Z value, by ID number, or by tag. When drawing the GameObjects, the GameLevel calls each of the GameObject's draw functions by order of their Z value, allowing for an arbitrary number of layers in a game level. The GameLevel also stores the viewport into the current level.

The most used class is the GameObject class. This is the class that the end user will inherit from, and has a series of functions such as onUpdate and onCollision that get called on specific conditions during the game loop are met.

The final parts of our engine are the various Asset classes. SpriteAssets are used to represent animated sprites, and can be scaled or rotated. ShapeAssets are similar to SpriteAssets except that they are defined by a set of parameters defining a shape, and not by a texture loaded from the disk. The SoundAsset lets the user play specific sound files, either looping or just once. TextAssets define a graphical text element that can be scaled, rotated, and colored in different ways. There is also a PhysicsComponent, which has a defined shape, velocity, and weight. The level updates all PhysicsComponents so that they move according to the laws of kinematics, and also register collisions when they occur.

Our prototype consists of a single game level, with a few objects to control the viewport position, the grid displaying, the various animations that occur during the game, and the python interface to Core Impact Pro. We have a Tower object that stores all of its relevant information from Core Impact Pro as it receives it, and it also stores all of its connected Towers. The Tower object calculates its position using an elastic spring network graph that continuously adjusts its position as the game updates, responding to new Towers by moving around to fit the new tower into the graph, until the graph reaches a state of equilibrium. Each link connecting towers is a separate game object that contains references to both connected towers, and provides a line for the line pulse object to move along.

To communicate with Core Impact Pro, we have a separate class that we fork off into its own thread. It initializes the python interpreter and loads the interface module. In order to send events to Core Impact Pro, we store the event XML in a static vector protected by a mutex. The python runner class checks the vector each iteration through its loop, and if it finds an event it sends it off. The runner also polls the python module for any new events that may have come back from Core Impact Pro. If it finds new events, it stores the returned XML in a separate mutex-protected vector that our event listener game object constantly checks. When the event listener finds a new event, it parses the XML and performs whatever updates are necessary in the game.

3. 6 Use Case Scenario

In order to effectively develop our prototype, we came up with an extensive use case covering all of the important actions a user might want to take. This let us prioritize which features we wanted to develop, as well as to determine what should be made easy to access.

The first action a user would take when starting with a blank workspace is to run an information gathering module on a specific network range. They would click on a tower, and then click scan. This would bring up a window with options for them to modify. After clicking on run, the user would wait for feedback to indicate that new towers were found. When the scan completes, the user would look at the towers they have discovered, and notice that one of them still has an unknown operating system. The user would click on the original tower and re-run the scan on that specific target, changing some of the options to try and discover more information. Once the second scan completes, the user would see that the tower has a Windows operating system. The user would click on the tower and see detailed information about the tower, such as open ports and running services. The user would have enough information to attempt an attack, so they would click on the original tower and click on attack to bring up the attack module options. The user would first try running the attack with the default options, so they would click Attack. There would be feedback to indicate that there is an attack in progress, and when that finishes, the user would see the attack was unsuccessful. The user would retry the attack module with different options. After the attacking animation finished, the user would see the tower change to indicate that a new agent has been deployed on that tower. The user then uses that agent to run a harvest module to try and gather information. There would be another animation to indicate that the harvest module is running. When the harvest finishes, the user would activate that agent by clicking on the infected tower and clicking activate. This would change the appearance of the tower again to indicate that the tower is now under direct control by the user. The user would then run a scan from the new tower to find a different network. Once that scan completes, the user would see several new towers appear that were not known from the original scanning point. The user would then attack one of the newly discovered towers via the intermediate tower that they gained control over. When the user sees the animation finish and sees that an agent was installed, the user would then activate the tower. The user would click on the tower and then on the shell button to open a remote shell on that tower, which the user could then use to remotely take control of the target machine.

3. 7 Asset Development and Integration

When we started our initial development in A term, we did not have an artist to produce assets for our prototype. In B Term, we brought Beth Hankel onto our team, working as an artist for a single term as a separate ISP. From the beginning, our art direction has been more of a digital, wireframe, neon style similar to *Tron* or *Hackers*. During the beginning of B term, we brainstormed with Beth to try and produce some concept sketches, and tried to refine our ideas for our visual style. Beth produced some concept art, and after discussing the results with the rest of the team, we settled on a specific art style for Beth to work with. Beth produced 3D models and rendered sprites from them. Our engine loads animated sprites from a 32 bit sprite sheet, so we provided the frame size and expected file type to Beth, and she produced sprites that we could correctly load into our prototype.

4. Evaluation

The following two sections present details of a usability study carried out on April 2nd and 3rd 2012. The purpose was to gain objective and subjective feedback from first-time users, as well as to compare the performance of the prototype interface to that of the existing commercial one. The first section outlines the method, and the second presents and analyses results from the study.

4. 1 Testing the Effectiveness of Game UX

Our hypothesis is that game interaction techniques can improve the usability of professional, non-game software. The interface prototype we built calls on graphical/spatial metaphors, animations, immediate on-demand information, and contextual affordances. The interface also makes use of sound more than the existing Core Impact Pro interface, though this aspect was deemphasized in favor of graphical developments at the request of our client.

In order to measure the effectiveness of our prototype, we carried out a small usability study. Participants were recruited by a convenience sample of 28 students, most of them undergraduates at WPI (one student was an English major at another school). Many were majors in Computer Science and/or Interactive Media and Game Development. Some (16) specified that they had taken courses related to computer networking. The average self-reported proficiency in computer networks was 1.76 on a Likert scale from 0 to 4. (One student did not answer this question.) By class year, the participant counts were 2, 5, 4, 16, and 1 for Freshmen, Sophomores, Juniors, and Seniors, and unknown, respectively.

Students scheduled times to participate in the study on one of two days in 30 minute session blocks. Sessions were assigned roughly evenly to Core Impact Pro or Game UX Prototype, so that everyone in a session could receive instructions at the same time via a 5 minute presentation at the start of the session. Actual testing took place on 4 identical laptops with mice and headphones, arranged such that screens could not be viewed by other participants. Sessions contained a maximum of 4 participants each. In all, 10 sessions were conducted, with 4 for Core Impact Pro and 6 for Game UX Prototype, resulting in 15 Core Impact Pro samples and 13 Game UX Prototype samples.

During the testing phase, participants followed a prescribed task list (provided by the client). Their goal was to search a virtual network of machine images running Windows and

Linux and collect certain pieces of information relevant to a penetration tester or a network security professional testing for legal compliance. Ultimately, they were to compromise one of the hosts and "pivot" from that host to view a new network.

The tasks and questions are reproduced in APPENDIX C. During the first Game UX Prototype session on day 1, it became apparent that one critical piece of information was missing from one of the questions. All subsequent sessions used a revised version of this question. (On day 1, the revision was by hand; on day 2, we printed a new version.) The questions in general were not difficult, but chosen to reflect what was currently possible in the prototype; the task was overall very similar to the original storyboard.

The subjective survey was to be completed in the last 5 minutes of the session. It included 15 questions (Likert scale) to gauge subjective levels of confusion, interest, ease of interaction, realism, and distraction, as well as an area for comments and suggestions.

4. 2 Analysis of Testing Results

Most sessions went by without major issues, but several minor malfunctions did occur. For one sample, Game UX Prototype had been launched incorrectly and was not able to return information for the last two questions. That participant's survey answers reflected less satisfaction with the experience and the interface, but the bulk of the task test was valid. Another participant neglected to look at the task questions until the very end of the session, so the task score is not representative; the survey score corresponds to an experience similar to that for the other participants, though not identical because less directed. A third participant misread the instructions, following the directions for the wrong group, and so was impeded from understanding them properly for most of the session. The subjective score for this participant (at

1) was the second lowest among all samples collected, though the task score was roughly average (at 9.5). Where not otherwise noted, we have used these samples.

The average task test score was 9.95 (standard deviation = 2.482) out of a maximum possible score of 13. Figure 9 displays the distribution of plotted scores on the task test. The maximum score was achieved by two Game UX Prototype users and one Core Impact Pro user. The average subjective survey score was 14.732 (standard deviation = 7.699) out of a theoretical maximum of 36. Figure 10 shows the plotted scores for the subjective survey. Negative scores are also possible, and one Core Impact Pro user's subjective survey had an overall negative score of -12; however, this was the only negative score in either group. The minimum possible score is -22, a consequence of including more questions phrased in positive terms than in negative terms. As both experimental groups received the same questions, this was relatively unlikely to bias the outcome.

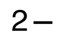


Figure 9 - Plotted scores for task test. The average score across both experimental groups is 9.95 (SD = 2.482), shown as a horizontal grey line.

-15 –

Figure 10 - Plotted scores for subjective survey. The average score across both experimental groups is 14.732 (SD = 7.699). The possible range of the test is [-22, 36], but only a single outlier is negative.

The average scores for the task test (n = 28) by experimental group were 11.00 (Game UX Prototype, n = 13, standard deviation: 2.73) and 9.04 (Core Impact Pro, n = 15, standard deviation: 1.72) out of a maximum possible score of 13 (achieved by one participant in the Game UX Prototype group). By two-tailed independent measures t-test, the probability of this difference appearing by chance is estimated at a p-value of 0.024. Given a typical significance level of α = 0.05, the improved task performance of the Game UX Prototype interface group can be considered significant.

The average scores for the subjective survey (n = 28) by experimental group were 16.92 (Game UX Prototype, n = 13, standard deviation: 3.75) and 12.50 (Core Impact Pro, n = 15, standard deviation: 9.55). By two-tailed independent measures t-test, the significance of this difference is estimated at a p-value of 0.091. Given a typical significance level of α = 0.05, the improved subjective experience of users in the Game UX Prototype interface group is not considered significant. The large standard deviation in scores for Core Impact Pro may reflect differences in background knowledge, interest, or some other factor.

We ran multiple-factor ANOVAs on both sets of test scores to compare the influence of self-declared network knowledge, student year, experimental group, and the interaction of network knowledge and experimental group. The degrees of freedom used in the calculations were 1, 3, 1, and 1, respectively. The associated least-squares means are shown in Figures 11 and 12.

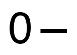


Figure 11 - Least-squares means for task test score by experimental group, with bars indicating standard error.

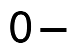


Figure 12 - Least-squares means for subjective survey score by experimental group, with bars indicating standard error.

For the task test, the F-ratios and corresponding p-values were as follows: self-declared network knowledge (F = 0.2479, p = 0.6240); student year (F = 3.8588, p = 0.0250); experimental group (F = 5.9209, p = 0.0245); and network knowledge * experimental group interaction (F = 0.0566, p = 0.8144). Thus, the strongest factor accounting for the difference in scores appears to be experimental group (p = 0.0245), with a smaller but equally probable effect from student year (p = 0.0250). Both of these effects are significant for α = 0.05, while neither self-declared network knowledge nor the network knowledge * experimental group interaction can be considered significant. The absence of an effect for self-declared network knowledge is interesting. A reasonable explanation is that the assigned tasks were chosen by our client not to be difficult.

For the subjective survey, F-ratios and p-values were as follows: self-declared network knowledge (F = 7.2667, p = 0.0139); student year (F = 3.5115, p = 0.0341); experimental group

(F = 6.0085, p = 0.0342); and network knowledge * experimental group interaction (F = 1.0873, p = 0.3095). The strongest factor in this case appears to be network knowledge (p = 0.0139), followed closely by experimental group (p = 0.0342). Both of these effects are significant for α = 0.05. Student year also meets the criterion for significance, though at a smaller effect size. The network knowledge * experimental group interaction by this calculation again cannot be considered significant. Notable in this set of results is that, unlike in the task test, where self-reported network knowledge had no effect, here it seems to be the most important factor accounting for differences in perception, regardless of which interface was used. This observation may be explained by the relatively large number of survey questions (11 of 13 used for scoring) about subjective levels of confusion, intimidation, ease and intuitiveness of use, and interest in the domain. The other two questions addressed the overall seriousness of the software and the realism of the presentation. (Note that two additional questions about graphics are not handled here, as participants did not fill these out consistently. See the full participant form in APPENDIX C.)

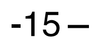


Figure 13 - Bivariate fit of subjective survey score to networking knowledge (R-squared = 0.1367).

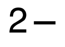


Figure 14 - Bivariate fit of task test score to networking knowledge (R-squared = 0.0127).

In summary, experimental group was a significant factor accounting for scores on both task test and subjective survey. In contrast, the network knowledge * experimental group interaction significant was not significant in either case. The reason that the ANOVA p-values suggest significance for both task test and subjective survey, whereas the t-test p-values suggest significance only for the task test, may be explained at least in part by the fact that the t-tests we ran were two-tailed, but ANOVA's calculation is one-tailed.

Running another ANOVA on task test scores, this time discarding the data from the user whose Game UX Prototype workspace had been started incorrectly and the Core Impact Pro user who chose not to follow the instructions closely (see the first paragraph of this section), produced values as follows: self-declared network knowledge (F = 0.6112, p = 0.4440); student year (F = 1.7107, p = 0.1987); experimental group (F = 6.3141, p = 0.0212); and network knowledge * experimental group interaction (F = 0.0457, p = 0.8330). In this case, experimental group was the only significant factor. The arithmetic means for the experimental groups were essentially unchanged at 9.48 for Core Impact Pro and 11.38 for Game UX Prototype. Least-squares means were 8.83 for Core Impact Pro (standard error = 0.638) and 10.77 for Game UX Prototype (standard error = 0.783). See Figure 15 below.

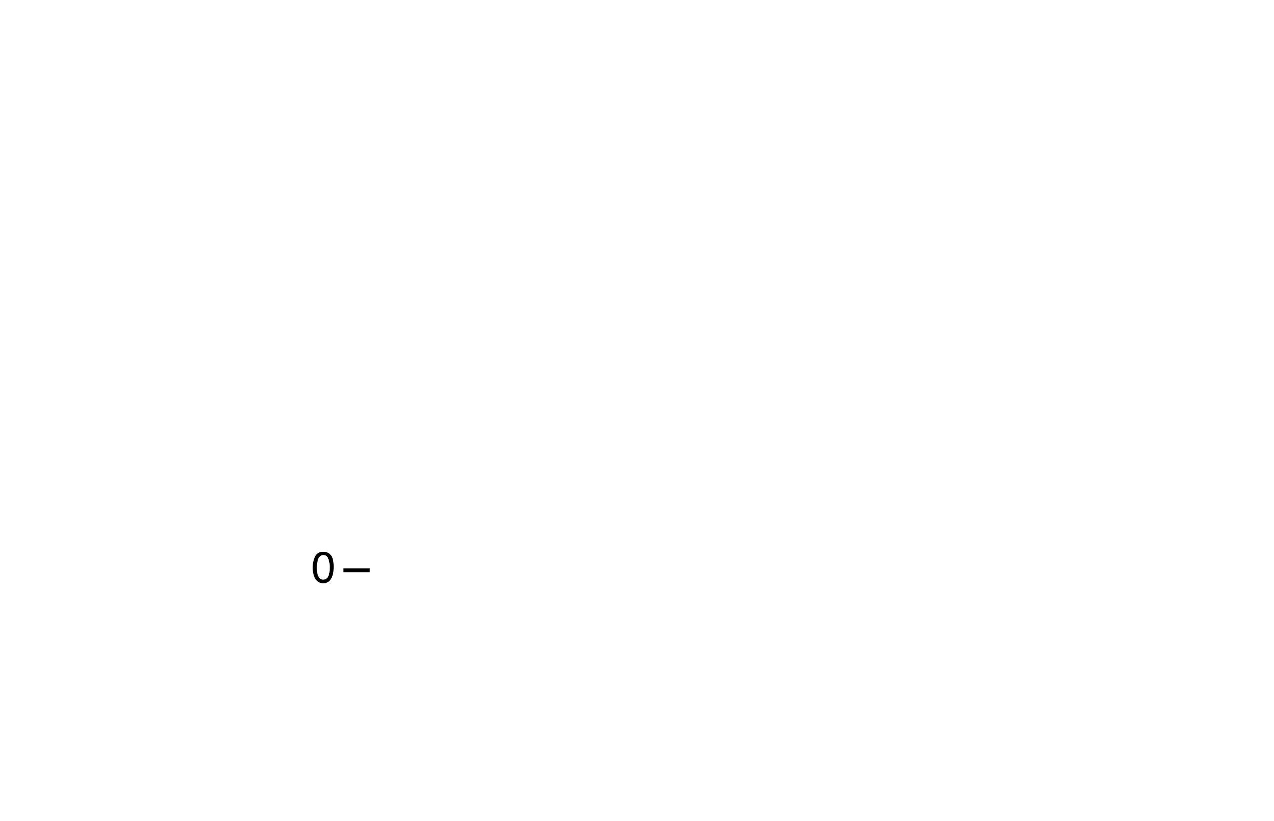


Figure 15 - This graph shows least squares means from re-running ANOVA on task test scores using a modified sample.

We also ran another ANOVA to re-examine the factors influencing subjective survey scores, this time discarding the data from the user who misread the directions, as well as that from the far outlier with the only negative score, with the following results: self-declared network knowledge (F = 1.5548, p = 0.2284); student year (F = 0.6732, p = 0.5796); experimental group (F = 1.0372, p = 0.3220); and network knowledge * experimental group interaction (F = 0.3891, p = 0.5406). The strongest contenders remain experimental group and networking knowledge, but without the two data samples that were omitted, the results are from a significance standpoint inconclusive, with no factor reaching a probability of statistical error $p < \alpha = 0.05$.

As the total number of participants was originally small (N = 28), another approach is to look at responses to individual questions. In particular, the prompt “The interface was intuitive” showed a significant effect, with better scores from the Game UX Prototype group and little to no influence from self-declared network knowledge (experimental group F = 4.3316, p = 0.0478; self-declared network knowledge F = 0.1123, p = 0.7404). Similarly, the response to "I understood the network and host representation" also showed a clear experimental effect (F = 8.6729, p = 0.0069) without a significant corresponding effect from self-declared network knowledge (F = 1.2780, p = 0.2690).

5. Conclusions

Core Impact Pro is a network security testing tool that is used to ensure the safety and security of computer networks. Core Impact Pro is a very powerful tool that can provide a large amount of information to its users about a network. This amount of information can be overwhelming for inexperienced users and some fail to utilize the more advanced features of the software. This was the major motivation for our project: to design a game-style interface that can be more approachable to inexperienced users.

Our team's project has received praise and positive comments from our peers, our professors, and the experts at Core Security. Our Core Security sponsor and his colleagues have enjoyed this project, and we have enjoyed working in collaboration with them. Exploring the techniques that games have always used to convey information and applying them to new domains has been an incredibly valuable learning experience. The project has demonstrated that complex tools can be simplified; the simplification enables understanding for a wider range of users and helps the user to more effectively utilize the tool.

Due to the large number of pieces of information in a game and the fact that there is limited screen space to display it all, game interfaces have to effectively prioritize information to try and maximize the usage of the available space. Our prototype split information about a tower into three separate tiers. The first tier consisted of information that we always displayed, such as whether or not a tower has agents installed, or what operating system the tower is running, or if the tower's agent is currently active is immediately visible. The next tier is comprised of things like the tower's IP address, full operating system name, and open ports. The third tier contains most of the rest of the known details about the tower, such as running services and potential vulnerabilities. Prioritizing the available info in this way allows the user to quickly determine the important details without becoming overwhelmed.

Game UX techniques can be applied to many other complex domains to improve user effectiveness. These techniques help to focus the user's attention on what's important at that particular moment, and also helps the user in prioritizing tasks. Game UX techniques also lend themselves well to reducing the amount of training necessary for new users to learn to use a given tool.

In order to test the effectiveness of our Game UX, we constructed an experiment. We recruited a group of individuals to test our interface and see how well they perform, compared to a control group doing the same test using Core Impact Pro. The participants were familiar with computer technology but had not seen either interface before the test. At the start of the session, the participants were given a brief introduction to the interface that they were using, along with a quick explanation of some of the terms we used. They were told to gather information on a pair of virtual networks hosted on their test machine. We provided them with a list of questions about specific tasks we wanted them to perform and asked them to answer as many of the questions as they could. The results of the experiment showed a significant increase in the effectiveness of users when using our Game UX. On average, users were able to answer two more questions using our interface, compared with using Core Impact Pro ($P = 0.\ 0225$). We also asked our participants subjective questions on how they felt about the interface that they were using, and our Game UX was rated 4 points higher than Core Impact Pro.

6. Future Work

Our team would love to see Core Security use our project. If this project is to be taken up by Core, they have a few choices ahead of them: “Should the game engine from the original project be used? Should the original project's code base be used as is, or reworked? Will the original project's design idea be utilized, or reconstructed?" Similar to the discussion that ensued during our group’s first few meetings (when we focused on preliminary questions regarding development), questions and discussion would be important during Core’s initial stages to improve their software.

Our team would also love to see future students continue the research and application of Game UX techniques that we began. Will other students at Worcester Polytechnic Institute see the advantages of working with this style of project? Will they see that games can be used to improve complex software? Will future students attempt other complex domains and utilize this approach to work on something outside of network security software?

6. 1 How Can Core Securities Continue this Project

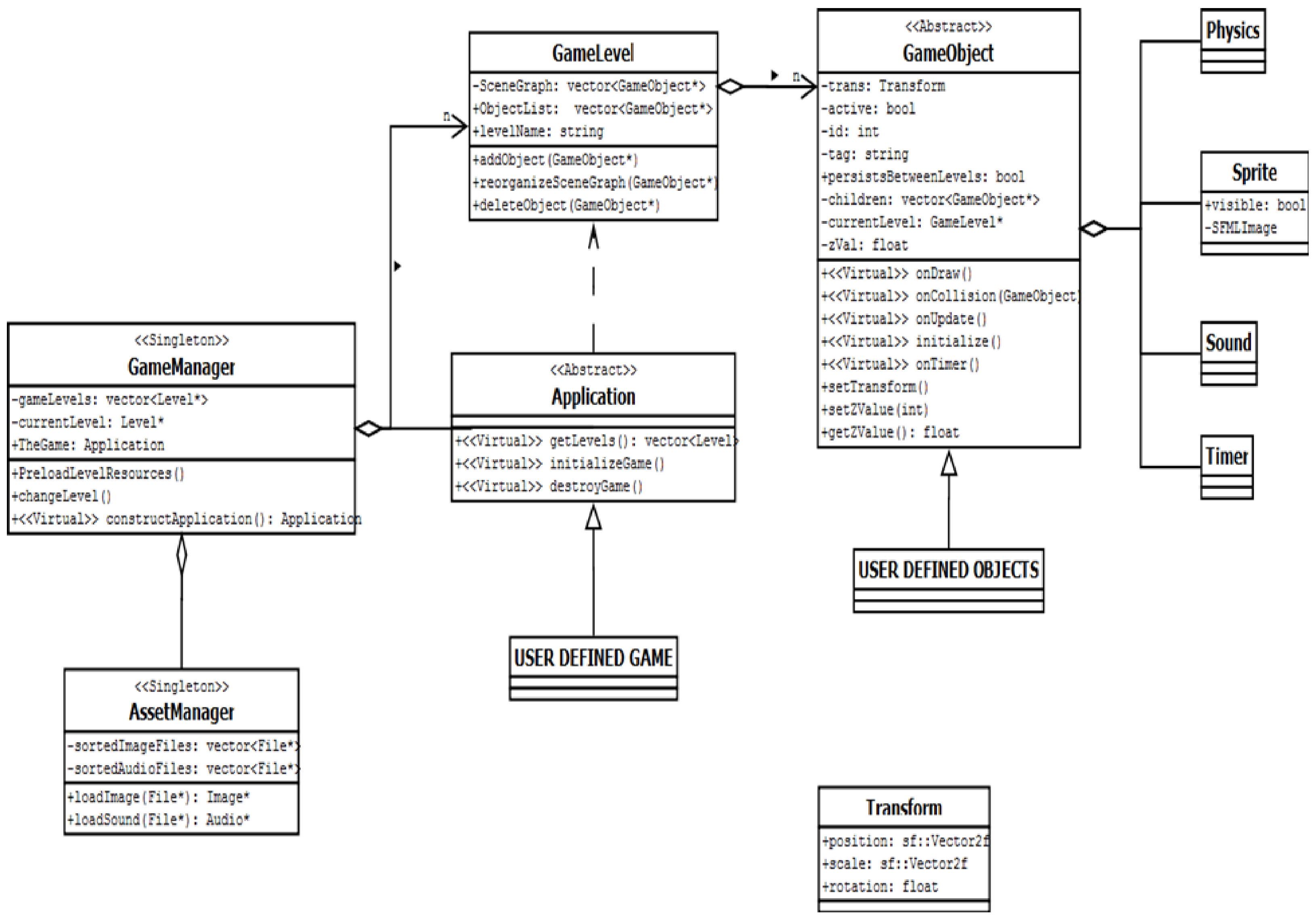
To continue this project, Core Securities will either continue working with our existing code base, or start a new project that applies the techniques we learned during our project. If Core is to continue with the existing code base, the developers at Core must learn the engine that our group has made and they must familiarize themselves with our existing code. If Core Securities decides to start a new code base, they will have to research suitable game engines to use for their project. We have provided a preliminary overview of potential game engines that Core Securities could use if they decide to start a new code base in Appendix D.

While we did finish all of the core features of our prototype, there are still several ways in which our prototype could be expanded on. Currently, our prototype only exposes a subset of the functionality available in Core Impact Pro, and ideally our prototype would provide the user with all the same functions. We would also create more badges to represent more of the important tower information in the first tier. Based on the feedback we received after presenting our project at Core Securities, we could also add ways to filter all of the visible towers so that the user can focus on a specific type of system. And finally, we would add more sound cues to our prototype.

The prototype currently only exposes some of the Rapid Penetration Tests available in Core Impact Pro. These tests run a large number of predefined modules with certain default parameters, modules that have been chosen because they are most commonly effective. Core Impact Pro also lets users run individual modules at their discretion, and we would like to allow our advanced users the same access to these individual modules. Core Impact Pro also provides reporting functionality which summarizes the output from the modules run, and this is another feature that we feel is important to include.

The other major feature we would have liked to implement is the ability to filter the towers based on specific parameters. Ideally, the user could have different layers for whichever parameters they find important, and then they could switch between the layers to see only what they consider to be relevant. Filters could also improve usability by allowing the user to highlight towers with specific attributes, so that they can better focus their attention on their particular goals.

Appendix A



<<Singleton>>
GameManager
-gameLevels: vector<Level*>
-currentLevel: Level*
+TheGame: Application
+PreloadLevelResources()
+changeLevel()
+<<Virtual>> constructApplication(): Application
<<Singleton>>
AssetManager
-sortedImageFiles: vector<File*>
-sortedAudioFiles: vector<File*>
+loadImage(File*): Image*
+loadSound(File*): Audio*
GameLevel
-SceneGraph: vector<GameObject*>
+ObjectList: vector<GameObject*>
+levelName: string
+addObject(GameObject*)
+reorganizeSceneGraph(GameObject*)
+deleteObject(GameObject*)
n
<<Abstract>>
Application
+<<Virtual>> getLevels(): vector<Level>
+<<Virtual>> initializeGame()
+<<Virtual>> destroyGame()
USER DEFINED GAME
<<Abstract>>
GameObject
-trans: Transform
-active: bool
-id: int
-tag: string
+persistsBetweenLevels: bool
-children: vector<GameObject*>
-currentLevel: GameLevel*
-zVal: float
+<<Virtual>> onDraw()
+<<Virtual>> onCollision(GameObject)
+<<Virtual>> onUpdate()
+<<Virtual>> initialize()
+<<Virtual>> onTimer()
+setTransform()
+setZValue(int)
+getZValue(): float
n
USER DEFINED OBJECTS
Physics
Sprite
+visible: bool
-SFMLImage
Sound
Timer
Transform
+position: sf::Vector2f
+scale: sf::Vector2f
+rotation: float

Appendix B

Engine/Utility Name	Language	Platform	License	Graphics	In-depth analysis	Sound	Networking	Scripting
Simple Fast Multmedia Language (SFML)	C++	Cross	zlib/png license (free)	2D	✓			
Simple DirectMedia Layer (SDL)	C++ (bindings to many other languages)	Cross	zlib license	2D/3D	✓			
AgateLib	.NET	Windows	Free	2D				
Agen	C++	Windows	Indie/Commercial	2D				
Allegro	C	Cross	Free(Open Source)	2D and 3D	✓	✓	✗	✗
Asphyre	Delphi/Delphi .NET	Windows	Free	2D/3D via DirectX				
Chingu	Ruby	windows/Mac/Linux	Free	2D				
ClanLib	C++	Widnows/Mac/Linux	Free(Open Source)	Accelerated 2D				
Cocos2D	Python, Objective-C	cross	MIT	2D				
CRM32Pro	C++	Windows/Linux	Closed Source; LGPL	2D via SDL/glSDL				
D'Enfent Engine	C++	Linux/Windows	GPL	2D/3D via OpenGL				
Daimonin	C (server), C++ (client), java (editor)	windows/Mac/Linux	GPL	2D/3D via SDL and OGRE3D				
DizzyAGE	C++	Windows	Free	2D via DirectX				
EasyWay Game Engine	Java	windows/Mac/Linux	opensource GPL	2D/3D via OpenGL				
Edge2d Engine	C++	Windows/Linux	Open Source	Library Independent				
Epee Engine	C++	windows/Mac/Linux	zlib/libpng	2D SDL				
Exult	C++		GPL	2D				
Flexible Isometric Free Engine (FIFE)	C++, Python	windows/Mac/Linux	Free/ open source (LGPL)	2D software renderer via SDL	✓	✓	✗	✓
Flixel	Actionscript	windows/other	MIT	2D	✓			
Game Maker	Delphi	Windows	Free and Commercial	2D/3D				
GameStart	C++	Windows/Wii	Commerical, Free trial	2D/3D				
Glint 3D Engine	LUA, C++	Windows/Mac	donation-based	2d/3D				
Golden T Game Engine	Java	windows/Mac/Linux	Free	2D via OpenGL				
Gosu	C++, Ruby	Cross	Free (MIT licensed)	2D via OpenGL				
Grail Adventure Game Engine	C++, Lua	Cross	GPL	2D via SDL				
Haaf's Game Engine (HGE)	C++	Windows	zlib/libpng	2D via DirectX				
Ika	C++	Windows/Linux	Free (GPL)	2D via OpenGL				
IwGame Engine	C++	ccross	Free	2D				
JEngine SSE	C++	Windows/Linux	Free (GPL)	2D via OpenGL				
Jgame	Java	Cross	Free (BSD)	2D				
Joge	Java	Cross	Free (Creative Commons)	2D via LWJGL				
JOGRE Engine	Java	Cross	Free (BSD)	2D				
Kobold2D	Objective-C, Lua	cross	MIT	2D				
Lavgine	C++, Lua	Windows	commercial	Hardware accelearted 2D				
LOVE	Lua	Windows/Linux	zlib/libpng	2D via OpenGL	✓	✓	✗	✓
Moai	C++, Lua	Cross	Open (CPAL)	2D via OpenGL				
MonoGame	C#	cross	Free	2D				
NetGore	C#	cross	Free	2D via SFML				
ORX	C/C++	cross	zlib	2D				
Playground SDK	C++, Lua	Windows/Mac	Free	2D				
Plib	C++	Cross	Free (LGPL)	2D				
Popcap FrameWrok	C++	Windows	Free	2D	✓	✓	✗	✗
PTK Engine	C++	windows/mac	Free and Commercial	2D				
PVLE Game Engine	C++	cross	GPL/ Proprietary	2D				
RPG Make 2003 or XP	C/Delphi	Windows	Shareware	2D				
Saq2D	C#	Windows	Free	2D via XNA				
sge2D	C		MIT	2D				
SLUDGE	C++	cross	Free (LGPL)	2D via OpenGL				
Stratagus	C		GPL	2D				
Thousand Parsec Framework	Python, C++, others	cross	Free (GPL)	2D/3D				
Torque 2D	C++	Cross	Commercial	2D	✓	✓	✓	✓
XNA	C#	Windows	Free	2D/3D	✓			

Appendix C

WPI – Gaming UX Usability Test – v6

Last saved: 4/26/2012 11:19:00 AM

Background:

Class:	Freshman	Sophomore	Junior	Senior	Grad

Major:

Networking Knowledge: (Little or None) 0 1 2 3 4 (Advanced)

Where did you learn about networks? (Course, Job, Personal experience?)

Test Machine: Date/Time:

Gather information:

- Gaming UX: run Scan from /localagent
- Impact PRO: run RPT: Network Information Gathering

Target IP range: 10.0.5.* Take all other defaults

After it completes answer these questions.

Note: Data will update until the scan completes.

Which host(s) have port 3389 open/listening?

Which host(s) have port 3372 open/listening?

Which hosts have filtered ports?

Determine the operating system of any hosts with unknown operating systems

Hint: If network information gathering scanned too few ports, you might miss discovering an OS. Retry.

Which host has an unknown operating system?

Were you able to find the operating system by following the hint?

Attack host 10.0.5.22

- Gaming UX: run Attack
- Impact PRO: run RPT: Attack and Penetrate

Target IP range: 10.0.5.22   Take all other defaults

If an attack is successful, CORE software deploys an agent onto the target host, and this allows the target host to be controlled remotely. Vulnerabilities that leave a host open to attack and exploitation have standard CVE (Common Vulnerabilities and Exposures) identifying codes.

Does 10.0.5.22 have any vulnerabilities?

Write down any CVE codes from the host.

Which exploit/vulnerability allowed the agent to be installed?

Hint: Find the agent.   See “deployed with” or “exploit”

Pivot

Pivot=use an exploited host to look deeper into the network.

Impact PRO:

- Set the agent deployed on 10.0.5.22 as the “source agent”
- Run RPT: Network Information Gathering
- Target IP range: 10.0.6.*

Gaming UX:

- Activate 10.0.5.22
- Run Scan from the exploited machine
- Target IP range: 10.0.6.*

What additional hosts were found?

The network testing experience was enjoyable (Not at All) 0 1 2 3 4 (Very Much)
I learned about network security (Not at All) 0 1 2 3 4 (Very Much)
The network testing experience was intimidating (Not at All) 0 1 2 3 4 (Very Much)

The interface was intuitive (Not at All) 0 1 2 3 4 (Very Much)
I would want to use this tool (Rarely) 0 1 2 3 4 (Frequently)
I am excited to learn more about computer security (Not at All) 0 1 2 3 4 (Very Much)
The activities were confusing (Not at All) 0 1 2 3 4 (Very Much)
I understood the network and host representation (Not at All) 0 1 2 3 4 (Very Much)
I knew what was going on at all times (Not at All) 0 1 2 3 4 (Very Much)

The network and host representation seemed realistic (Not at All) 0 1 2 3 4 (Very Much)
The graphics were N/A (Distracting) 0 1 2 3 4 (Helpful)
The sound was N/A (Distracting) 0 1 2 3 4 (Helpful)
The software seemed serious (Not at All) 0 1 2 3 4 (Very Much)
Network security is boring (Not at All) 0 1 2 3 4 (Very Much)
Where would you place the difficulty of effectively using this tool—on a scale from 1 (very easy) to 10 (very difficult)?

Please explain (briefly) any difficulties you had. Record any other comments or suggestions here as well.

Appendix D

Choosing the right engine is a lengthy process. There are hundreds of engines that exist in the world that are not proprietary (non-proprietary engines can be used by any developer as needed). There are commercial engines (engines requiring a paid license) and open source engines (free engines). Various types of engines in each of these categories have individual benefits and drawbacks. A lot of research should go into the choice of engine. A developer should consider the types of games the engine is best suited for, the quality of the documentation and the community, and the license that the engine requires for use.

An example of our team's research on the higher-level engine material is illustrated in Appendix B. The information presented in Appendix B is a brief summary of the engines we researched, providing an overview of the functionalities the engines provide. Our team has provided more detailed research into eleven of the engines listed in Appendix B. We researched the websites of these eleven engines, and information was gathered on each engine and its framework.

Simple and Fast Multimedia Library (SFML):

According to the SFML website, SFML is an easy to use multimedia API written in C++. It is the modern, object-oriented alternative to Simple DirectMedia Layer (Which is detailed later in this paper). SFML is composed of packages to suit the developer's needs. The developer can use SFML as a minimal windowing system to interface with OpenGL, or as a fully-featured multimedia library for making games or interactive products. [1]

[1] http://www. sfml-dev. org/features. php Visited 3/25/2012 SFML website

SFML has windowing and GUI frameworks, such as GWEN, that are specifically built to work with it, making the creation of Game UI's easy and understandable. These add-ons add more to the library. SFML is not an actual engine, but its groups of libraries make 2D game creation simpler.

SFML has windowing and GUI frameworks, such as GWEN, that are specifically built to work with it, making the creation of Game UI's easy and understandable. These add-ons provide more to the libraries capabilities. SFML is not an actual engine, but its groups of libraries make 2D game creation simpler.

The community for SFML is also great, the creator of the libraries, Laurent Gomila, actively participates in the forums. Gomila is ready to respond to any and all questions, he will also respond in a quick fashion most of the time. The user base is very helpful as well and will readily help out in the forums with any pleas from new and experienced users.

SFML is available in the following languages:

- C++
- C
- . Net (C#, VB. Net, C++/CLI, . . .)
- Python
- D
- Ruby

SFML handles graphics, audio and some networking functionality. It is cross platform as well, which means that it will work on Linux, Mac and Windows, as long as they have Open GL. SFML is easy to learn, it is simple and fast.

<u>Simple DirectMedia Layer (SDL)</u>:

According to the SDL website, SDL is a cross-platform multimedia library designed to provide low level access to audio, keyboard, mouse, joystick, 3D and 2D via OpenGL. It is used by multiple companies and multiple platforms to take advantage of the current hardware, it has been used on such projects like the award winning Linux port of "Civilization: Call To Power. "

SDL supports a large variety of operating systems including the major three of Linux, Window and Mac. The code contains support for gaming systems and little known operating systems such as AmigaOS, Dreamcast, Atari, AIX, OSF/Tru64, RISC OS, SymbianOS, and OS/2, but these are not officially supported.

SDL is written in C, but works with C++, and has bindings to several other languages, including and not limited to C#, Java, Lisp, Lua, Objective C, Perl, PHP, Python, Ruby.

SDL is distributed under GNU LGPL version 2 license. This license allows for free use of SDL in commercial programs as long as the developer links with the dynamic library. [2]

The SDL site explains what its frameworks and libraries have done and what they have the potential to do. SDL is a very popular and versatile library. Well known games, such as *Second Life* and *World of Goo,* have been made with SDL. SDL has an active community and its forums contain almost 30, 000 posts.

Allegro:

Indicated on the Allegro website, Allegro, mainly Allegro 4 and 5, are cross-platform libraries aimed at video game and multimedia projects. Allegro handles low-level tasks such as creating window and the basic game loop. The game loop consists of getting user input, drawing

[2]http://www. libsdl. org/ visited 3/25/2012 SDL website

to the window, and playing sounds. Allegro 4 and 5 generally abstract away the underlying platform. Allegro is not a game engine: it is a collection of multimedia libraries and its power does not lie solely in game making. [3]

Allegro 5 is used to take advantage of modern hardware and operating systems. It only supports 2D graphics primitives natively. Allegro can work alongside 3D and is made to be modular. Its user forums are strong and active, having over 420, 000 posts. Allegro's web page is also set up to be very navigable; and each section is easy to research and find.

Flexible Isometric Free Engine (FIFE):

FIFE is a cross platform 2D game creation framework written in C++ with Python bindings. FIFE is designed to work in the development of all 2D game types but its major focus is 2D isometric game creation. [4]

FIFE is a newer engine; its forums are still posted to, just not on a regular basis. FIFE also has another website up and running: fifeengine. net. This site is more in-depth and has all the documentation needed for a beginning user to become a literate user of the tool.

Flixel:

The Flixel site states that Flixel is an open source game-making library completely free for personal or commercial use. It is written in Actionscript 3, typically a language used for flash development. Flixel is designed to be used with free development tools. Flixel is easy to learn, extend and customize. Flixel also boasts that it has been used in hundreds of games, including

[3]http://alleg. sourceforge. net/readme. html visited 3/25/2012 Allegros Sourceforge
[4]http://sourceforge. net/projects/fife/ visited 3/25/2012 FIFEs sourceforge

IGF nominees, Adult Swim games, and avant-garde experiments. Flixel is a development tool that numerous users engage with for their very first games. [5]

Flixel is a common and widely used engine. It is intended for flash-style game development and is programmed using Actionscript rather than C++. Flixel's website has multiple video tutorials and a highly active forum. The forum includes multiple experienced users, like the creators, who help out new users with questions and concerns. Flixel also is well documented and has a very easily navigable website. Flixel is a good web based engine. If a user has a preference for Javascript, then Jixel (a HTML5 engine that is based upon the Flixel framework) would be their preference of engine.

LÖVE:

LÖVE is a framework for making 2D games utilizing Lua. LÖVE is free, and its licensing allows for use in any style of project that the user desires. It has been used in open source projects, free games, and for-sale games. [6]

LÖVE has an in depth wiki with pages upon pages of documentation. Its forums are widely used with thousands of posts continuing the discussion of the product and the capabilities that it has. The community and the documentation for LÖVE are lively and up-to-date on the site. Lua, the language that LÖVE uses, making LÖVE a scripting based engine. LÖVE is an engine that supports multithreading and sound and physics, but does not support networking that has to be created as a separate component by the user of the engine.

PopCap Framework:

[5]http://flixel. org/ visited 3/25/2012 Flixels homepage
[6]https://love2d. org/wiki/Main_Page visited 3/25/2012LÖVEs wiki page

The sourceforge for the framework states that the PopCap game framework, also named SexyApp Framework, is a flexible high-level library. SexyApp provides functions and reusable components that make the development of the games made from utilizing this system quick and simple. It is designed to facilitate rapid development for high-quality games, such as PopCap's “Bejewled” and Sandlot Games' “Cake Mania. ”[7]

PopCap Framework is open source and has an 83% approval rating. PopCap Framework has roughly 80 downloads a week--it is an active site with a user base that consistently works on the framework with fellow developers. With a large number of downloads per week, it is clear that there are many new users interested in learning about the product. A major downfall for the framework is that the community, tutorials, and documentation are hard to find. The sourceforge is easy to locate, but other important aspects require searching. Some developers have developed wrappers around this framework to create more focused libraries.

Torque2D:

Torque2Ds licenses have to be purchased for use of the product. Some engines may require royalties from released products or be completely free, but Torque2D requires a fee.

According to Torque2D's “garage games” site, Torque 2D is an extremely powerful and easy to use 2D game engine. It is built using the Torque architecture and therefore offers many features that the Torque3D game engine has. Torque2D comes with a powerful Integrated Development Environment (IDE) that allows for creation and placement of game objects into a

[7]http://sourceforge. net/projects/popcapframework/ visited 3/25/2012 PopCap Framework Wiki

visual representation of the world without any need of programming. Purchasing the Torque2D license gives the owner Torque’s complete feature set. [8]

The feature set comes with a level editor so that objects can visually be placed and assigned values through an Integrated Development Environment (IDE). Torque2D has in-depth and detailed documentation and tutorials to help users work with the engine. Its community is large, with multiple and diverse posts on the forums. Torque2D is still actively and widely used.

XNA:

XNA is a set of libraries made by Microsoft. The best development environment to utilize XNA would be Microsoft Visual Studio. XNA and Visual Studio are both Microsoft products and made to work with one another for game development. XNA has easy integration with Xbox360 and can also be used with the Kinect through way of the Kinect sdk, which has been released by Microsoft. Easy integration allows for the product to spread into multiple media forms besides just the PC. XNA has a highly active user base and is very popular among developers. It is Microsoft owned and free to use. XNA is not an engine in a “true sense, ” it is a library like SFML and SDL. Regardless, it proves very useful. With Visual Studio 11s IDE, XNA is an extremely powerful tool that is well documented and easy to use.

Mayhem:

Mayhem is our team's engine. It has what is essential of engines--physics, scene graph, game object support, level management, audio, and sprite integration. Mayhem is created for 2D games (the essentials previously mentioned are the necessities of 2D games). Although Mayhem

[8]http://www. garagegames. com/products/torque-2d visited 3/25/2012 Torque2D webpage

is a 2D-specific engine, it has a “3rd” dimension: the scene can have a layer of sprites so that the drawing and proper collisions can be tested based on a “Z” axis. The Mayhem engine can have its frames per second (fps) set to what the users want. The Mayhem engine is made for mono sound (not stereo or 5. 1).

7. References

[1] Mudge, Raphael. *Armitage*. Computer software. *Armitage - Cyber Attack Management for Metasploit*. Web. 25 Apr. 2012. <http://www.fastandeasyhacking.com/>.

[2] Jordan, Craig, Matt Knapp, and Dan Mitchell. *CounterMeasures - An Interactive Game for Security Training*. Tech. no. MQP-MLC-SG10. WPI. Web. 25 Apr. 2012.

[3] "Go Hack or Go Home: CSAW Winners Choose Victory." *Polytechnic Institute of New York University*. Web. 25 Apr. 2012. <http://www.poly.edu/news/2011/11/15/go-hack-or-go-home-csaw-winners-choose-victory>.

[4] "BackTrack Linux: The Ultimate Hacker's Arsenal." - *ADMIN*. Web. 25 Apr. 2012. <http://www.admin-magazine.com/Articles/BackTrack-Linux-The-Ultimate-Hacker-s-Arsenal>.

[5] "Welcome to the National Collegiate Cyber Defense Competition." *Welcome to the National Collegiate Cyber Defense Competition*. Web. 25 Apr. 2012. <http://www.nationalccdc.org/>.

[6] Mejia, Robin. "Red Team Versus Blue Team: How to Run an Effective Simulation." *CSO Online - Security and Risk*. CSO Online, 25 Mar. 2008. Web. 25 Apr. 2012. <http://www.csoonline.com/article/221695/red-team-versus-blue-team-how-to-run-an-effective-simulation>.

[7] Jelena Mirkovic, Peter Reiher, Christos Papadopoulos, Alefiya Hussain, Marla Shepard, Michael Berg, and Robert Jung. “Testing a Collaborative DDoS Defense In a Red Team/Blue Team Exercise." *IEEE Trans. Comput.* 57.8 (2008): 1098-1112. Web. Web. 25 Apr. 2012.

[8] "CSSIA Midwest Collegiate Cyber Defense Competition (MWCCDC)." *CSSIA NSF ATE Center*. Moraine Valley Community College. Web. 25 Apr. 2012. <http://www.cssia.org/ccdc/>.