“CYBER RTS”
PROJECT 400 REPORT

Stephen Dunne (s00130835)
Hons. BSc Computing in Software Development

1 TABLE OF CONTENTS

2 INTRODUCTION

2.1 BACKGROUND

Growing up I always enjoyed PC gaming, my favourite genre to play was always Real-Time Strategy (RTS) games from the late 90's such as Age of Empires and Red Alert. It was my love for this genre and my big interest in Object-Orientated programming that led me to the idea for my final-year project being born.

2.2 WHAT IS A REAL-TIME STRATEGY (RTS) GAME?

In an RTS game, the player must control a civilization/ town/ army that can be expanded upon by resource-gathering. There is typically certain units and structures dedicated to this task, and gathering these resources allows the player to build new units and structures, thus increasing the size and power of their civilization throughout the course of the game.

The main objective for the player of the game is to build their base to strong enough power that they can attack and destroy any enemy target/s on the map. A typical RTS game also includes technological development, wherein completing research tasks increases the units and structures stats, as well as unlocking new units and structures for purchase, and new attacks to use on the enemy [1]

2.3 WHAT IS THE GAME?

The game developed is a prototype RTS game with some unique aspects to it.

One of the unique feature of my game will be the setting. While most RTS games have a physical war setting such as space, WW2 or the medieval ages, mine will be physical-digital. The setting is inside a PC where the player must build their motherboard (mother base) and must develop components such as Hard-drive (More Units), CPU (Technology), GPU (Battle Units), RAM (Research) and Networking (Ability to attack) cards. Using the combined efforts of all these components, the player will be able to hack and destroys another computer (AI base) controlled by a hacker that wants to take down government database servers.

Resources will come in the form of Bitcoins which must be harvested to build up your base while connecting to electrical outlets will give the user more power to charge all of the components built.

The game will also include a mobile companion application that can be used to gather more experience-points (XP, used to level up the base), as well as view user stats. The application can also be used to create your team's flag, which can then be exported to the PC game.

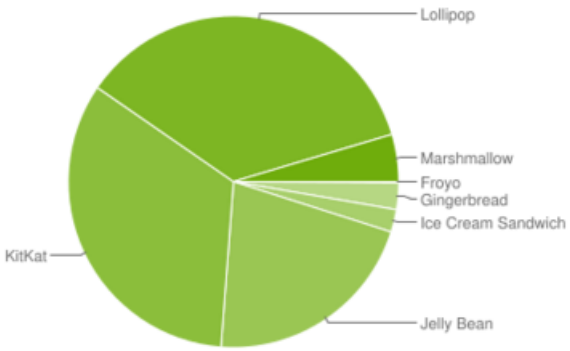
2.4 THE PROTOTYPE ITSELF

The game will be Windows PC exclusive, developed on the Unity Games Development platform. I decided to go the Windows route over say Android/iPhone as the game itself includes a lot of text, text-based objects (buttons) and panels. I wanted to be sure the game could be easily enjoyed by anybody who ran it on a computer with a monitor of resolution 800x600 or higher.

The companion app will be an Android application, and is also developed in Unity. The application will require Android Jelly Bean Version 4.1 at the very least. I set this as the base target as at least 95.1% of all current Android users are running this version or higher, while the other 4.9% are using Android Ice Cream Sandwich Version 4.0.4 or lower. As such, I feel V.4.1x is a fair cut-off point. [2]

An SQL database hosted on a server, paired with a simple log-in system (only requiring an email and password) will be used to sync the data from the mobile application to PC game, and vice versa.

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	2.6%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	2.2%
4.1.x	Jelly Bean	16	7.8%
4.2.x		17	10.5%
4.3		18	3.0%
4.4	KitKat	19	33.4%
5.0	Lollipop	21	16.4%
5.1		22	19.4%
6.0	Marshmallow	23	4.6%



Pie-chart showing the percentages of different Android Versions used

3 GOAL AND SCOPE

3.1 GOAL

I had a number of goals that should be achievable by any user who plays my game project prototype. The goals are listed as such below.

3.1.1 Main Game

A user starts with a certain amount of currency and must end up building a computer powerful enough to take down, through virus' and hacks, a very powerful enemies PC that is planning on taking down the Governments servers and leaking all every citizen's personal information.

With the limited amount of resources available at the start, the user will only be able to build a basic PC. But through the ability to start harvesting bitcoins (the games currency) and finding additional power sources, the user can buy more PC components and upgrades for these components, while also being able to provide power to them.

Having a more powerful PC means tasks become faster to complete, in turn increasing the users total XP. When a certain XP target is met, the user can spend bitcoin to level up the base. Doing this opens up new research options.

Before being able to attack the enemy, the user must research and build a Network Card. It is building this Network card that allows the user to go "online" and research more powerful virus' (such as "Trojan Horses") as well as send any weaker virus' previously researched to the enemies' PC. The user can also use the Network Card to find out personal information on the enemy; such as finding out just how powerful their computer is. Obtaining this information gives the user an idea of how prepared they are to attack the enemy, as they only have one chance at it as well as only 30 days (in-game time, 30 minutes in real-time) to complete the mission.

However, using the Network Card to find out information on the enemy will alert said enemy to the players' presence. The enemy can now attack the player at any time so it is important for the player to set up in-game defense protocols. This is done through purchasing VPN's.

Obtaining VPN's will mask the players IP address while performing any research tasks completed through the Network Card. Masking the IP address means the enemy will end up sending attacks to the wrong "dummy" tasks. If the enemy manages to get past all the VPN's and attack your base, it's game over for the player.

3.1.2 Companion App

Before starting the game, a user can choose to login with a username and password or just play as a guest. Logging into the game gives the user the ability to use the companion app alongside the game.

Within this Android application (also developed in Unity) a user can log in with the same credentials as on their PC to view in-game stats on the fly. These stats include Username, XP, total power, bitcoin collected and a host of different variables associated with the users' in-game PC components.

The companion app also includes a pair-matching mini-game which can be used to gain more experience-points (XP). The user is shown two terminals and must find the duplicate code between them within a time-limit to earn XP. Clearing a terminal, starts a new round as well as giving the user some bonus time and an XP-multiplier. After each game the app sends the XP earned to the main-game where it is added to the users' XP total, which is used to level up their base when it reaches a certain target.

3.2 SCOPE

The scope of this game is moderately large as it will be programmed in C# and JavaScript on the Unity Games Development Platform which allows for quick development. After performing some preliminary research and having built several games projects and prototypes within Unity in the past, I feel I will be able to accomplish the following:

3.2.1 Display Panels

I expect to include multiple panel displays shown at all times within the game, these are as follows:

1. **Game Panel**: This panel is the window to your game world where you can see all of the components of your computer in operation.
2. **Taskbar Panel**: Options such as Exit, Save, Load etc. will be included here.
3. **Main Panel**: This panel includes a few key sections to the game. Player info such as name, money, XP, power and power requirements are displayed up top. Next is the main store. Here a user buys computer components such as CPU's, GPU's, RAM. The bottom of this panel displays the players current Level and option to level up when it's available.
4. **Information and Notification Panel**: On the left hand side of this panel is where information on a selected computer component is displayed, as well as any upgrades available for said component. The right hand side of the panel shows any notifications for the player such as a particular upgrade being finished. Time remaining in the game is also shown.
5. **Notification Window**: Alerts will be posted here that show if any "building" or "soldier" is under attack. It will also alert the player if resources are running low.

3.2.2 Gameplay Features

Using the panels above, I expect to be able to include these features within the scope of the game:

1. **Build A PC**: Buy Motherboard, CPU, GPU, RAM, Network Card etc. and see them displayed in the game panel
2. **Level-Up System**: Research new hardware to earn XP and level up your base.
3. **Path Finding**: Clicking somewhere on a map with units selected sends them there while avoiding obstacles. (ended up being removed)
4. **Base Attacking**: Ability to attack AI enemy bases while also being able to get attacked by them.
5. **Research Abilities:** Research new virus' for use against the enemy.

3.2.3 Companion App

The scope of the companion app will include these features:

1. **Login System:** Logging in is what pairs the application to your PC profile
2. **Pair-Matching Mini-Game:** The mini-game that will be included to increase player XP
3. **Stats:** View stats from user profile on the fly. (only a proof of concept in final version)

3.2.4 Basic Features

However, certain aspects of the game will be quite basic while other features may be left out entirely. These include:

1. **Simple Audio**: To be able to focus on the programming side of this game, I will have very little audio, if any. Audio included will be a backing track and simple clips played when attacking, resource collecting, expanding the base, movement and notifications. (Not ended up being included)
2. **Basic Enemy AI**: As it stands, the enemy AI is going to be quite simple. The enemies PC will be a set size from the start of the game and the player must expand their own base and destroy the enemies before a timer runs out. If the timer runs out, it's game over as the enemy has successfully destroyed the governments network database.
3. **No Random Map Generation**: I would have liked to be able to have random map generation in this game but with time as limited as it is, I don't feel that this will be accomplishable. Time permitting, I may work up a prototype of this for the project demonstration. (Not ended up being included)
4. **Saving and Loading**: If I can get a companion app to connect with the game through a server-hosted SQL database, I also hope to include a saving and loading option within the game.
5. **Highscore System:** Again, providing I get an SQL database set up I will include a Highscore system that allows users to compare Fastest victory, highest level and XP against friends and other users world-wide. (Not ended up being included)
6. **(Companion App) User-Icon maker:** I would also like to include an icon-making mini-game in the application. Here, the user will be given a 32x32 grid that they draw a custom-design into. When they're completed, the design is sent to the main-game and displayed alongside the users' name. This section isn't vital and will only be done provided I have the time and resources.

4 Main Areas Of Research

4.1 Platform for Project Development: Unity or MonoGame

There are many differences between Unity and MonoGame when it comes to games development. Unity is a fully-fledged games engine that already provides physics, graphics, assets, shaders, particle systems, and more whilst MonoGame is a framework that implements Microsoft XNA 4.0.

Unity and MonoGame can be just as good as each other when it comes to certain projects as both of them allow the use of C# Scripting and are great for developing 2D games.

4.1.1 Benefits of Unity

Unity being a high level tool focused engine and as such, would provide me with a list of easily accessible features. If you want high-level tooling support and an engine that does more for you, but at the cost of code flexibility, Unity is a great choice.

While MonoGame can be just as good, if not better, than Unity at developing 2D games, Unity is much more powerful when it comes to 3D games development.

Unity also has a huge array of features when it comes to UI development.

4.1.2 Benefits of MonoGame

MonoGame is a low-level code-focused framework and would allow me to implement just the features I need. If you prefer maximum flexibility and are more the code it from scratch type, MonoGame is an excellent choice. However, using MonoGame means the games developer must implement all of the features desired through libraries, whereas Unity provides them by default. [3]

4.1.3 Choosing Unity

For my project, I ended up deciding on using Unity for several reasons. These are as follows:

1. **Familiarity with the IDE:** Having previously used Unity for my final year BSc in Computing with Games Development project, I'm well acquainted with its wide array of features and limitations.
2. **Intuitive:** I also find it extremely intuitive as it allows me to quickly build the prototype of my game without me needing to previously develop a base engine- as would be the case in MonoGame.
3. **Game Scene Display Screen:** Unity also shows me my game-scene in an on-screen window as I develop it. I can drag and drop new GameObject's straight into this scene and see them in the world instantly. Modifying any of the objects variables such as translations, sprites, material within the Unity IDE updates them in the game-scene display on the fly.

4. **The advantage at developing 3D games:** Unity has much more native support for developing 3D games. This was a major influence on my decision as my original game concept was to include both 3D and 2D GameObject's in a 2.5D style setting.
5. **Built in UI development features:** My game, being an RTS, incorporates a lot of UI elements. As such, another major influence for going with Unity was because it provides me with so many UI development features from the get-go.

However, while I ended up deciding on using Unity as the games development platform for my game, I did still use the Mono IDE for all scripting purposes.

4.2 NETWORKING REQUIREMENTS FOR MOBILE COMPANION APP

For this project, I needed a networking system to be able to synchronize data between the main game and mobile companion application. I done research into several different potential solutions for this and have listed them below:

4.2.1 Photon Unity Networking [5]

This was the first potential solution I looked into as it is what was previously used in my 3rd year final project "Jet-Ski" joyride. It was a project team-mate who took care of incorporating this plugin to all for all networking features into that game, but seeing how it worked successfully I decided to look into it myself for this project.

Photon Unity Networking (PUN) is a plugin for Unity that implements Unity's own networking API on top of Photon. It allows cross-platform support while also taking care of the back-end, allowing the developer to focus more on the games front-end programming. The free-version of PUN even gives you 20 dedicated servers for free to use within the game.

4.2.2 BitRave

Having a student free-trial with Azure, I looked into possibly using that with Unity to provide synchronization between the game and mobile application. After some research, I found the Unity BitRave plugin.

This plugin incorporates the Azure Mobile Services, which is a networking back-end ran as a service that supports multi-platform development. The plugin itself is designed to work on every platform that Unity supports, including Android, and allows your unity game to use the Azure cloud back-end for synchronization between devices and platforms. [6][7]

4.2.3 Mono/.Net framework

The third possible route I researched about is scripting through Mono/.Net as unity Supports this. Mono/.Net has a lot of different built-in networking tools, which can be used for creating lots of different network connections. This route is what took most of my attention so I have included all my points of research on how to set up Networking in this way below.

Going down this route would involve a few steps to ensure a working, full-proof solution. First, the main game should host up a network service that the mobile app connects to as a client.

For sending the companion application the players' stats, the service must offer a simple method called something like "RecievePlayerStats". This method can be used to send the client any player stats you want to specify in any format: Typically json is considered the best.

Ensuring the Network Service is discoverable is what allows the client to connect to and read info from the game. This network service can be rolled out by yourself through Mono/.Net's built in socket service tools like UdpClient[8]

Include a button in the game which starts the synchronization process (Label the button with something like "Pair") with the mobile companion application. When the button is selected, a thread is spun up,

acting as the server. It is important for the threaded and so there are multiple tools which can be used to help with this. The one I looked into the most is Thread Ninja as it has a small footprint on the CPU usage. [9]

Once the thread is spun up, a UdpClient is created that is used to start attempting to “retrieve” the companion application. It is important to have a timeout on this so the socket can be closed after a certain time duration if not request ends up being receive, thus saving on processor usage on tasks that aren’t being completed. However, the user may click the pair button before even opening the companion application, so it’s also important to ensure the timeout is long enough to accommodate this.

Within the companion app itself, there should also be a pair button. Pressing this, creates a second UdpClient that also broadcasts a request signal with timeout.

If the game host receives the request within the timeout period, it will gather up the IP address and desired port of the client mobile companion app, while also sending out a request that includes its own IP address and desired port. Pairing is complete once the companion app receives and saves that data.

Now using a while loop on the PC Game, you repeatedly open the Receive for the EndPoint mobile app, which is the EndPoint. On the mobile app, you also have a while loop that continuously tries to connect to the server and request the users’ stats information. It is in this request that the mobile app calls the “RecievePlayerStats” method to receive the required player data.

Once the host receives the message, it will parse out the command and respond by the sending back the players’ stats data.

The reverse of the above can also be done if you want to send some information from the companion app to the main game, such as sending XP earned in the mini-game towards the players’ levelling up.

In between requests, the client should be allowed to rest for a short duration of time, every 200 milliseconds or so. [10]

4.3 SIMILAR TITLES DEVELOPED IN THE PAST

I done a lot of research into RTS titles of the past before getting to work on my game. The research I done on these games helped me to further plan out:

1. How I would make an intuitive and clean user-interface.
2. How to make the different components of the in-game PC unique from one another
3. How to make the game more balanced and user-friendly for first time players

Researching how these games looked and played also gave me some ideas on how to keep my own prototype unique in its style and format. Two of the games I looked at were as follows.

4.3.1 Age of Empires

One of the most popular RTS games ever made and being my biggest inspiration for Cyber-RTS, I looked into what exactly made it so loved. I found that a lot of it had to do with its research and resource-gathering system. This system allowed users to have different playstyles from one another by giving them the ability to research into and then upgrade individual types of buildings using resources gathered.

Another one of its reasons for being so popular was its clean UI and intuitive design. The game has a a 2.5D isometric gameplay format.

Age of Empires Gamescreen

4.3.2 Red Alert

Just as, if not more popular than Age of Empires, Red Alert and its successors are one of the most widely loved RTS game series ever made. Created in the 90's by Westwood Studios (Now EA), Red Alert was extremely popular for its aesthetics and Game-play mechanics. The most excellently developed mechanic of which is the base-building system.

A player starts with just a "House of Operations" building, from which they gradually build up the rest of the base and army using minerals and gold they gather from silos around the map. I researched what made each of these building unique in terms of attributes and what new research options having them would give you.

Red Alert 2 Gamescreen

4.4 How to make a Successful User Interface

Expanding on the research into what made Age of Empire and Red Alert's UI successful, I looked into clever user interface development so I could make mine as clear and intuitive as possible for users.

Previous to this project, I had never developed any piece of software or game with anywhere near such a complex UI. As such, I had to do a lot of research and fact finding on what makes a successful user interface. What I found was eight key characteristics a good user interface needs. These characteristics are: [11]

1. **Clear:** The most vital characteristic of a successful design is its clarity. A user needs to be able to quickly figure out how the use the UI or they may get frustrated and fed up, causing them to stay away from your product in the future.

2. **Concise:** With large applications, you don't want to be throwing too much information at a user as it can become overwhelming. As such it is important to keep things clear as well as concise. If you can factor the explanation of a feature in a single sentence instead of multiple, that's exactly what you should do. While keeping your UI both clear and concise can take a lot of time and effort to achieve, the end product will always be much greater.

3. **Familiar:** Familiarity is the most important characteristic for making an intuitive UI design. Intuitiveness in a UI is when something is instinctively understood and comprehended and familiarity is when you see something that reminds you of something you've previously encountered. Creating an intuitive design involves identifying things a user is familiar with and associating them with commands that are similar. An example of this is using a "Floppy-Disk" icon to represent a save-file button.

4. **Responsive:** The responsiveness of an interface is how fast it loads and updates. Laggy and slow interfaces are a sure fire way to ensure a user will get fed up with your system. Seeing the contents of the system load and update quickly greatly improves the user experience.

5. **Consistent:** An interface should have a certain level of consistency throughout. Having a consistent interface builds usage patterns and associates familiarity in users. A user will be able to quickly learn what user cases new buttons, tabs and icons are associated with based on previously extrapolated experiences.

6. **Attractive:** While it is more important to make your UI clear, concise and responsive, it is also a great benefit to make it attractive so to create a much higher level of satisfaction in users. It is important to style the fashion and look of the UI for your target market, as its their satisfaction you're relying on to have them tell their friends and family about your product.

7. **Efficient:** Efficiency is a massive factor in making a UI successful. It is an efficient UI that allows users to perform functions fast and with little effort. Finding out how to make your software efficient involves putting time into working out what exactly a user will want to

typically achieve, and then setting up the UI in a way that lets them accomplish their target in as few clicks as possible.

8. **Forgiving:** Users make mistakes all the time, they may select a button or tab that they didn't intend on by accident. Sometimes these buttons may cause something to be deleted or a textbox to be cleared, so it is the duty of the UI developer to ensure protocols are in place to deal with this. This may come in the form of an undo button, or a dialog box asking the user are they sure they want to do perform the task associated with the button they clicked. There is nothing more frustrating than not being able to undo something being deleted by an accidental click, so having these "undo" procedures in place gives a level of relief to an otherwise distressed user.

5 Design and User Interface

5.1 Design

My game prototype has a simple top-down 2D style while also incorporating particle-effects to add some flavor and style. The design of the game strictly follows a “cyber” theme.

5.1.1 Fonts

To keep consistent within the “cyber” theme of the game, I wanted to use fonts on the UI that had a technological-feel to them. Realizing that none of the default Windows fonts had the aesthetic I was looking for, I found some free fonts online that much better suited the games theme. These are as follows:

I felt that these fonts much more accurately suited the theme of my game over anything already provided, and thus incorporated them into the different UI elements of the game.

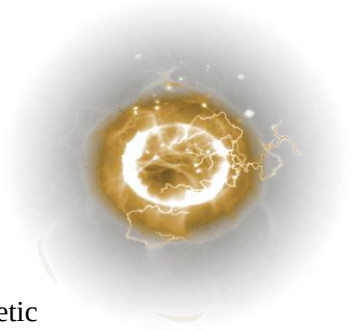
5.1.2 Sprites

Again, wanting to stick with to the "cyber" aesthetic of the game, I decided to both; design sprites myself and modify some sprites found from online sources. The sprites follow this same cyber style and are used in many different areas of the game. Some of these areas include panel backgrounds, button backgrounds, unique sprites for the different components of the in-game PC components. Some of these sprites are shown below.

Button Background sprite:

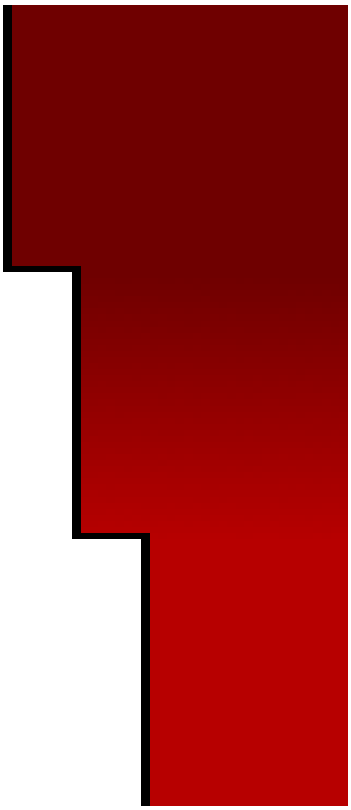
A sprite used as a button background within the game. For every button, the "white" space can be modified in Unity to any RGBA color.

The "charge-ball" sprite:



I used this sprite for power-sources and bit-coin mines. For the bit-coin mines, I used Unity to modify the hue to a deep purple. However, unlike typical sprites within the game, this sprite isn't used as a still image. It is used for particle-effect emissions, being shown with many of the same sprite except all in slightly different sizes, rotation speeds and rotation directions. This creates a digital-vortex style effect that I feel suits the aesthetic of the game.

Main Panel Background Sprite:



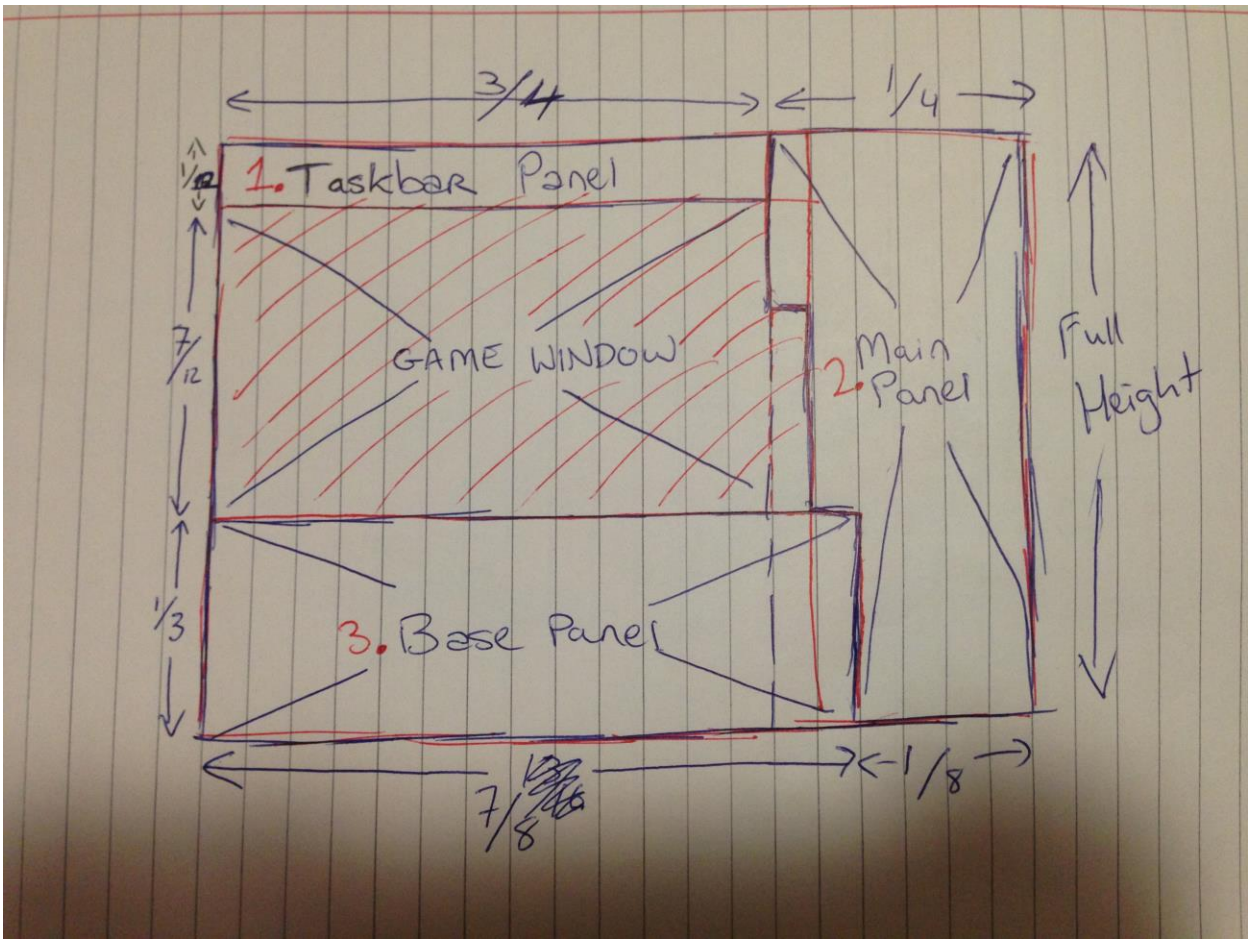
This is one a sprite used for the background of the right-hand game panel. For all game panel objects, I used background sprites similar to this as it has a blocky feel to it, which when over-laid with different buttons, labels and displays of interest, suits the "cyber" style of the game.

5.1.3 Audio-Files

The prototype game will use 8-bit .wav files for its audio files. I chose 8-bit files as they most accurately suit the "cyber" theme I am going for. Different Beep and Click sounds will take up the majority of these files as I want the user to feel like they're actually listening to the sounds a computer makes while booting up.

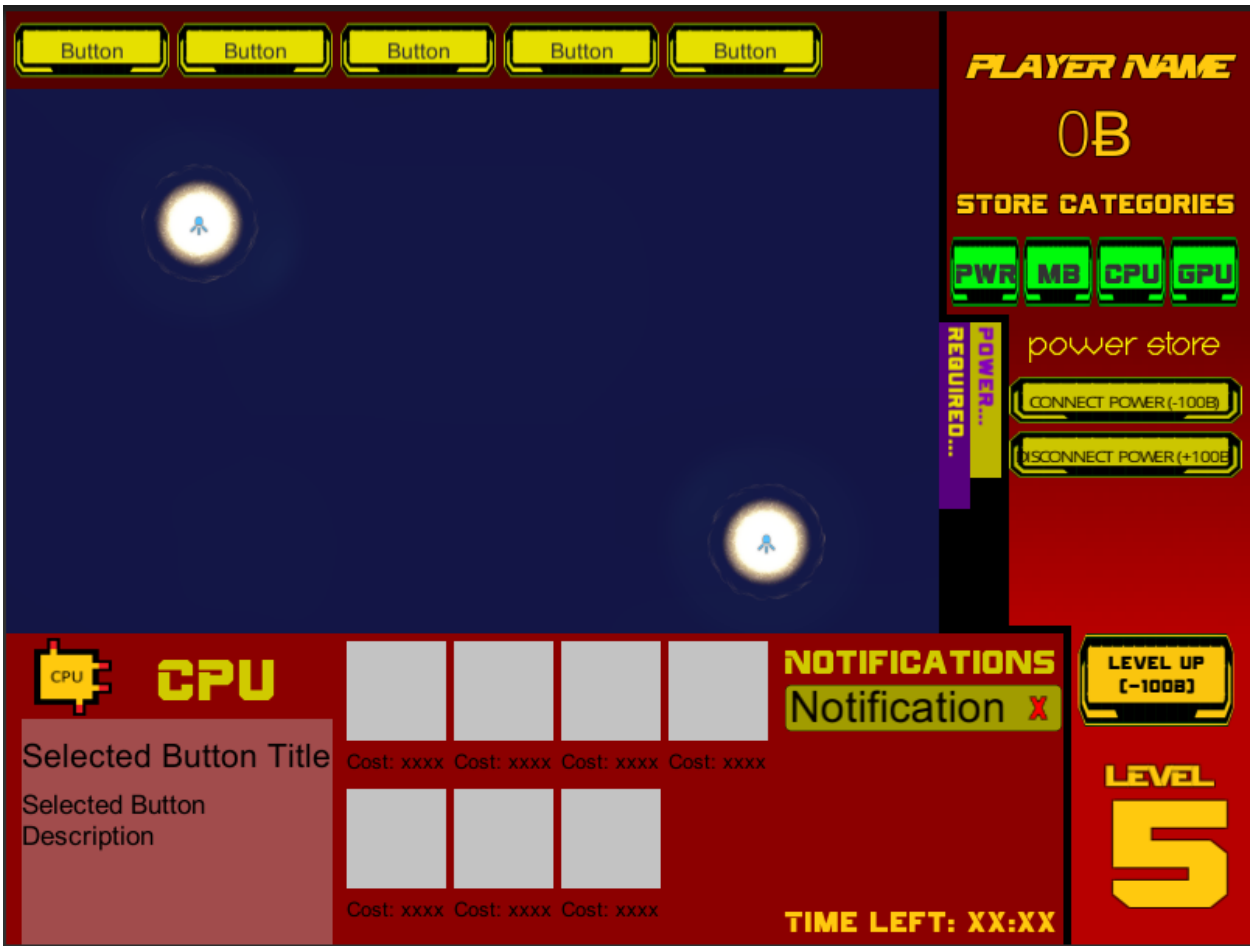
5.2 USER INTERFACE

Making an RTS game, the User Interface is an extremely important aspect towards making my prototype run and look like a quality product. I spent a good deal of research learning about good UI design and that helped me work out the following template for how the User Interface should be displayed on screen.



As you can see, the game screen is divided into four different sections. Three of sections used make up the Taskbar, Main and Base panels of the User Interface, whereas the fourth section is the game window that shows the player their base.

One of the first things I done for the prototype in Unity was set up the UI. The prototypes initial panel set-up can be seen below.



This is the game prototype User Interface panel set up. As you can see, I also have a lot of the different buttons and labels set-up with their backgrounds and font-styling too. This is near enough how the final game prototype ended up looking bar a few exceptions. Ex. Not including the current XP and XP target underneath the users' current level (which is displayed in the bottom right.

Here are some descriptions of what each panel contains:

5.2.1 Taskbar Panel (Top-Bar)

This panel is used just for file-associated buttons such as Save, Load, Pair and Exit.

5.2.2 Main Panel (Right-hand Bar)

From top to bottom, this panel contains:

1. Players username
2. Player Current Bit-Coin count
3. Store-Categories Title

4. Buttons for each of the Store Categories
5. (Purple Bar with "Required" written lengthways on it) Current power level required to operate all the in-game PC components
6. (Yellow Bar with "Power" written lengthways on it) Current power level. This yellow bar must be longer than the purple one for the users' PC components to all get enough charge.
7. Selected Store Category Title (currently "power store")
8. Buttons associated with the Selected Store Category. ("Connect Power" and "Disconnect Power" are both attributes associated with the "Power Store")
9. Level-Up button (This only becomes available when XP targets are met).
10. Current Level
11. Not Included: Current XP and XP Target to Level-Up

5.2.3 Base Panel (Bottom Bar)

This panel is divided up into three different segments. Left-hand side, center and right-hand side. The contents of each of these segments are listed below.

Left-hand side segment:

The contents of this segment change dynamically depending on what PC component you've last clicked on within the game window. You are shown the components sprite to the top left, followed by the name of the component. Underneath you are given some details about your selected component such as speed, level, etc.

Center segment:

Research options are shown here, and just like with the left-hand side panel segment, these options change dynamically based on what PC component is selected from the game window. Available research tasks are shown with a white background, and hovering over them displays their information in the Left-hand side segments "description" text box. Completed research tasks become green in color, while currently unavailable research tasks are red.

Right-hand side segment:

The final segment of the Base Panel is divided into two key areas.

One is the Notification tray. Here is where the user will be shown anything of importance such as research tasks being completed, enough XP gained to level-up, etc. Notifications can also be closed by clicking the red cross icon at the end of them.

The second area is the "Time Remaining" label. Here the user is shown how long they have left to complete the game objective.

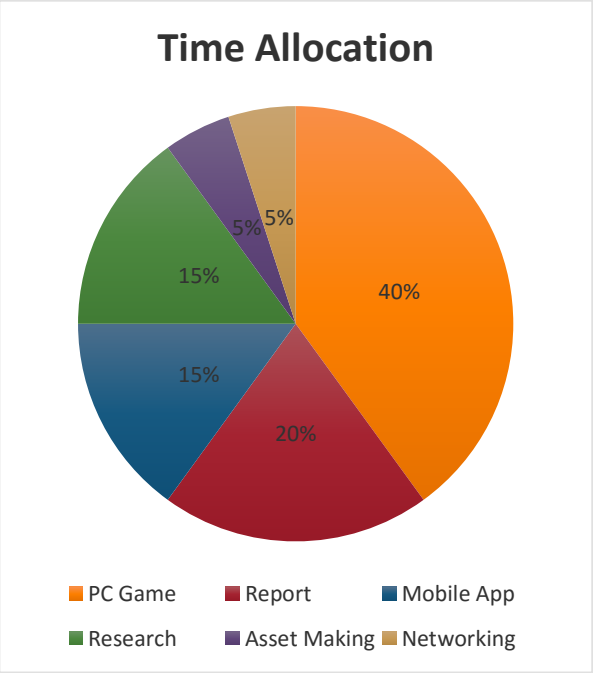
6 TIME MANAGEMENT

6.1 TIME ALLOCATION

For the development of the prototype, I tried to allocate a certain percentage of time-spent into different area. The time was divided up as such:

PC Game (40%)

As the PC game prototype is the largest part of my project I would try to dedicate around 30% of all project development-time working on it. Time spent here would be writing C# for use in the game prefabs and other game objects and how they interact with one another. Setting up the UI, such as multiple game displays (stores, information panel, taskbar etc.) would also be worked on during this time.



Writing the Report (20%)

The report being such a vital component of the project deliverables, I tried to spend at least a fifth of all project-time working on it.

Mobile Companion App (15%)

I would spend about 15% of my time working on the mobile companion application for my game. Time spent here would be designing the UI

Research and Planning (15%)

Research for the project, while being an important aspect, doesn't actually require a huge amount of dedicated time so I felt 15% of all project-time was a fair figure. This figure is shared with planning time. Planning being time spent working out the User Interface and Design of the game, working out a MoSCoW diagram to find out what features Must, Should, Could and Won't make it into the final prototype.

Asset Making (5%)

I had planned from the get-go that every sprite in the game would be custom designed, so I felt that 5% of all project time was more than enough to get everything I needed without chomping too much into time needed for other, more key areas of the development cycle.

Setting up Network Component (5%)

The Networking Component being only a tiny aspect of the prototype of the main project, I allocated 5% of my time to it. Seeing as it was non-fundamental to the main goals of my application, I didn't want to spend too much time on the networking, especially if it was to end up being too difficult and time-intensive to incorporate.

6.2 MILESTONES

Throughout the development of my prototype, I would work out a road-map of milestones that I wanted to have completed within the next week, month, and entire duration of the project. These milestones would include such things as a feature being successfully added to the game, a certain area of my report being completed, a new set of game assets being produced, etc.

I kept these milestones on the wall behind my desk and seeing the deadlines come close would encourage me to try and have them completed on time.

The original milestone targets are listed below.

6.2.1 Original milestones prediction.

These are the milestones I hoped to have completed for my project by the start of each month. Initially wrote up in September with information added to each month as I further established what the game should include.

- **October:** Concept of game idea established
- **November:** Preliminary research. This research includes stuff like what IDE I would use for the primary development of the game and research into past game titles that are similar to my own game
- **December:** Main research (Lit Review) and Goals and Scope of the game. Which features I expect to have in the game. These features are established through the Lit-Review, finding out which ones are feasible and within my skill-level.
- **January:** First prototype of the game to be developed for first presentation.
- **February:** Re-Evaluation of my game based on feedback from first prototypes presentation.
- **March:** Project report and second prototype at least 30% completed (Re-doing the project from scratch). Have work on Database for pairing with mobile application started. Base building, enemy base and store should be finished by this point. Assets for game sprites should be designed at this point.
- **April:** Report and prototype deliverables at least 70% completed. Have the mobile application fully completed bar the pairing with PC. UI should be completed as well as upgrades for individual components.
- **May:** Bug testing, mobile application, Final game prototype and project report completed and ready for submission.

6.2.2 Monthly milestones by the week.

Below, I run through what my milestones were each week for the next month. Also, I didn't start the weekly milestones until the start of February. At that point I already had the initial prototype of the game created, as well as the report started.

February (After doing the presentation)

- **Week 1:** Re-evaluate game design and think about adjustments
- **Week 2:** Start developing Prototype V2.
- **Week 3:** Have base-building operational as well as store.
- **Week 4:** Have enemy base set up and at least 3000 words done on report.

March

- **Week 1:** Have UI set up (Panels and windows). Object selection fully operational for displaying research tasks available.
- **Week 2:** Mobile application under development. Main menu should be prepared at the very least. Work out resolution issues within the main game.
- **Week 3:** Have mini-game for mobile app up and running. Work out issues with android not displaying panels correctly.
- **Week 4:** Have database fully set up for the game-synchronization. (Couldn't be done so left out) 6000 words done on report.

April

- **Week 1:** Upgrades and research tasks fully established. Taskbar should be populated and operational.
- **Week 2:** Notification panel and enemy attacks operational.
- **Week 3:** Begin bug testing and final small features added to the game.
- **Week 4:** Bug-testing completed. Game and mobile app fully running. Report completed and ready for submission.

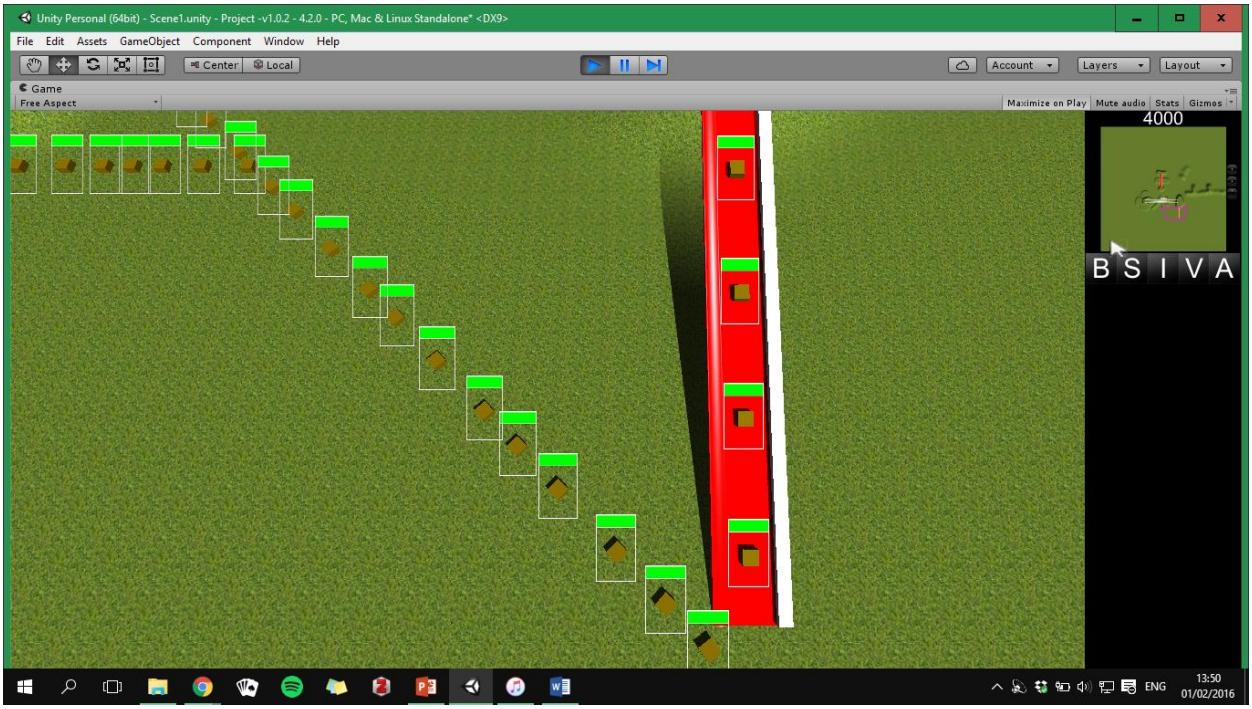
7 PROBLEMS ENCOUNTERED

Throughout the course of developing Cyber-RTS, I encountered my fair share of problems. I have listed some of the key problems below.

7.1 ORIGINAL PROTOTYPE LACKED CLEAR VISION

Originally, my game had a very different style and play format. Instead of the game having its current 2D style and management-system style gameplay, it had a very physical style, incorporating 3D models that represented all the individual parts of the computer components in a "Real-Life" style. What I mean by this is that parts to the computer such as the GPU and CPU would be represented as "buildings" while the hacks used by the player to destroy the enemies' PC would be represented as "tanks" which the player could send around the map to attack the enemy "buildings".

The tanks used A* pathfinding to navigate around the map while also avoiding other tanks and buildings. The screenshot below shows that original prototype in action.

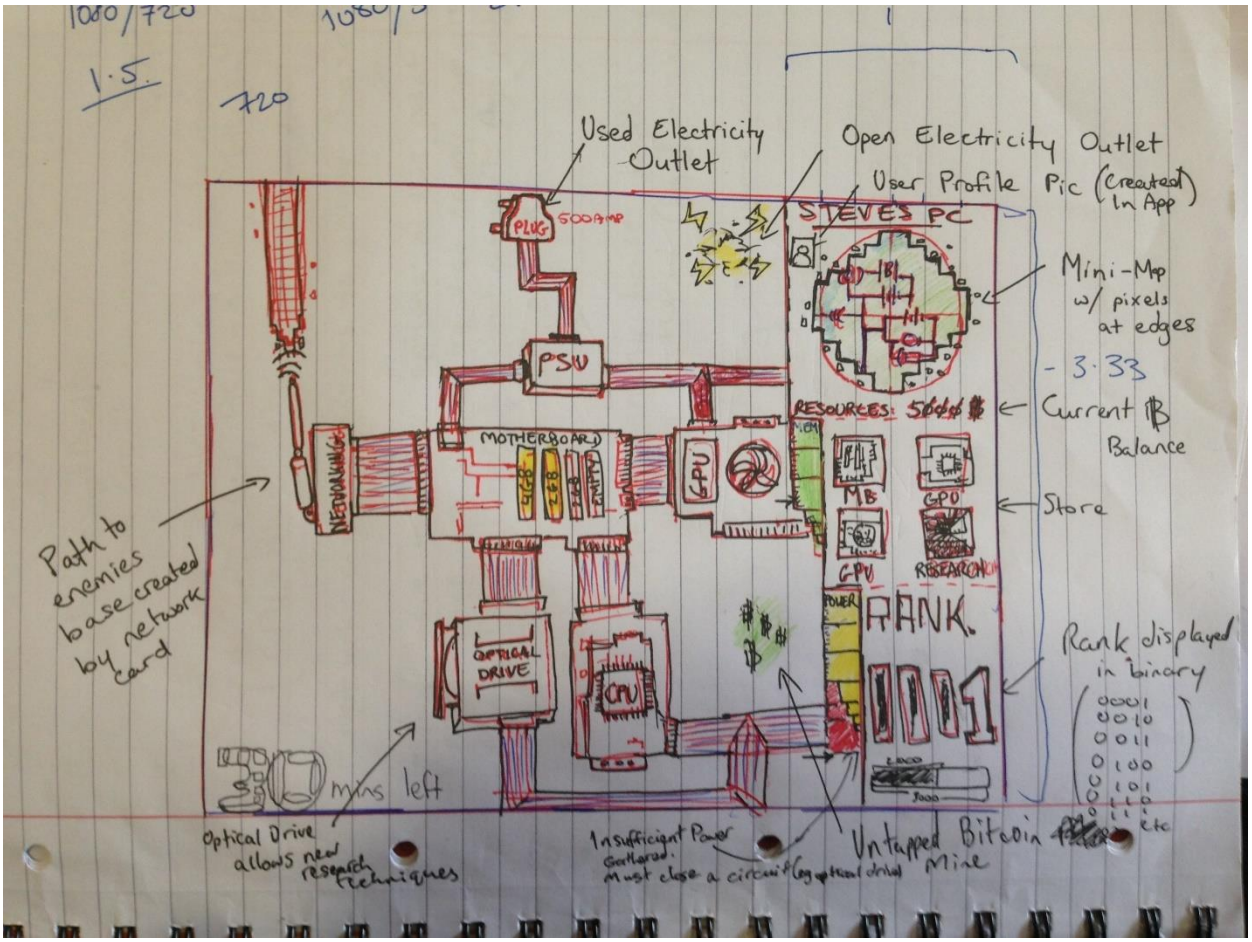


"Tanks" using A Pathfinding to navigate to the bottom of a slope and move up it towards their target location (note how they don't collide with one another)*

During the original presentation of this game prototype at the end of January, I was asked why I was going for a "cyber-digital" theme when I was representing all the individual components as tanks and

buildings as its presentation didn't really seem to match this theme of "cyber-digital" I was going for. I really thought about this question, and so after the presentation I ended up going back to the drawing-board to completely redesign the presentation of the game.

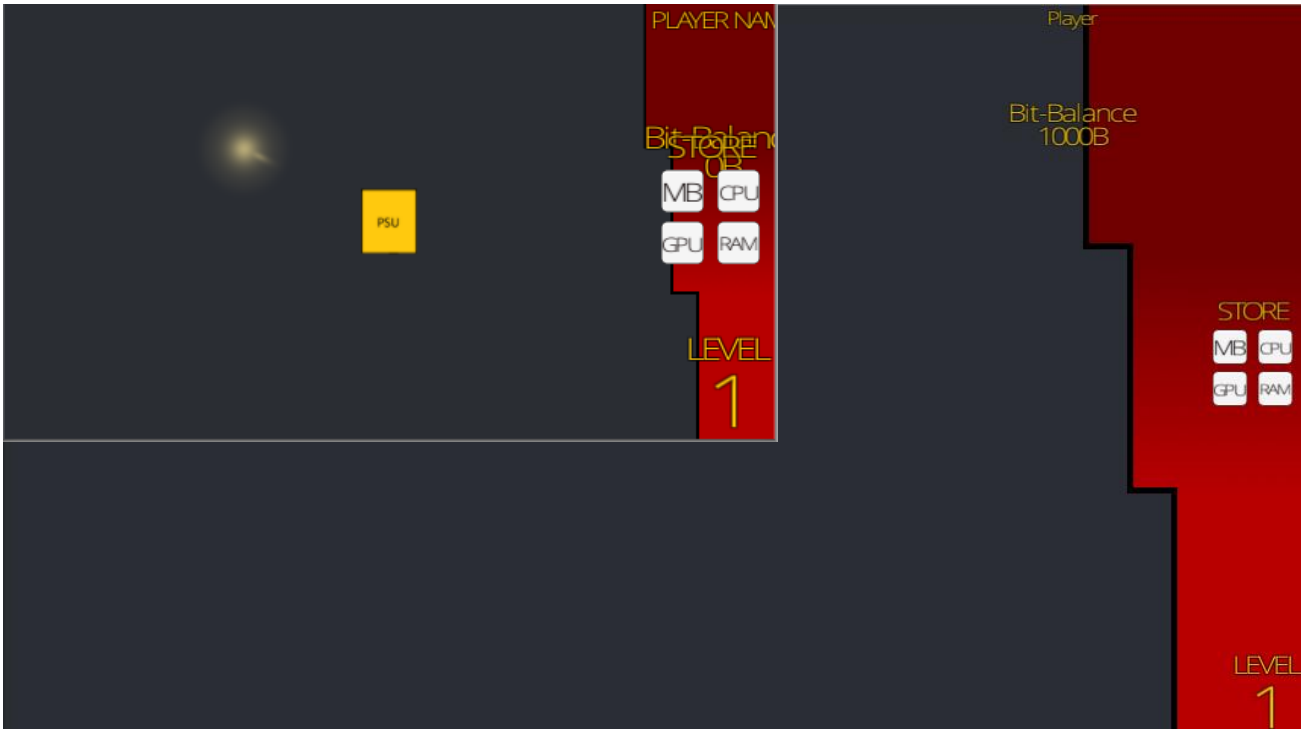
This eventually led to its current style as I realized the original-style I had in mind didn't even revolve around watching tanks blow up buildings, but instead it was a more management-style system where the game-display was more just a window into giving the player an idea of just how powerful their computer is.



The original re-design drawing.

7.2 RESOLUTION SCALING ISSUES

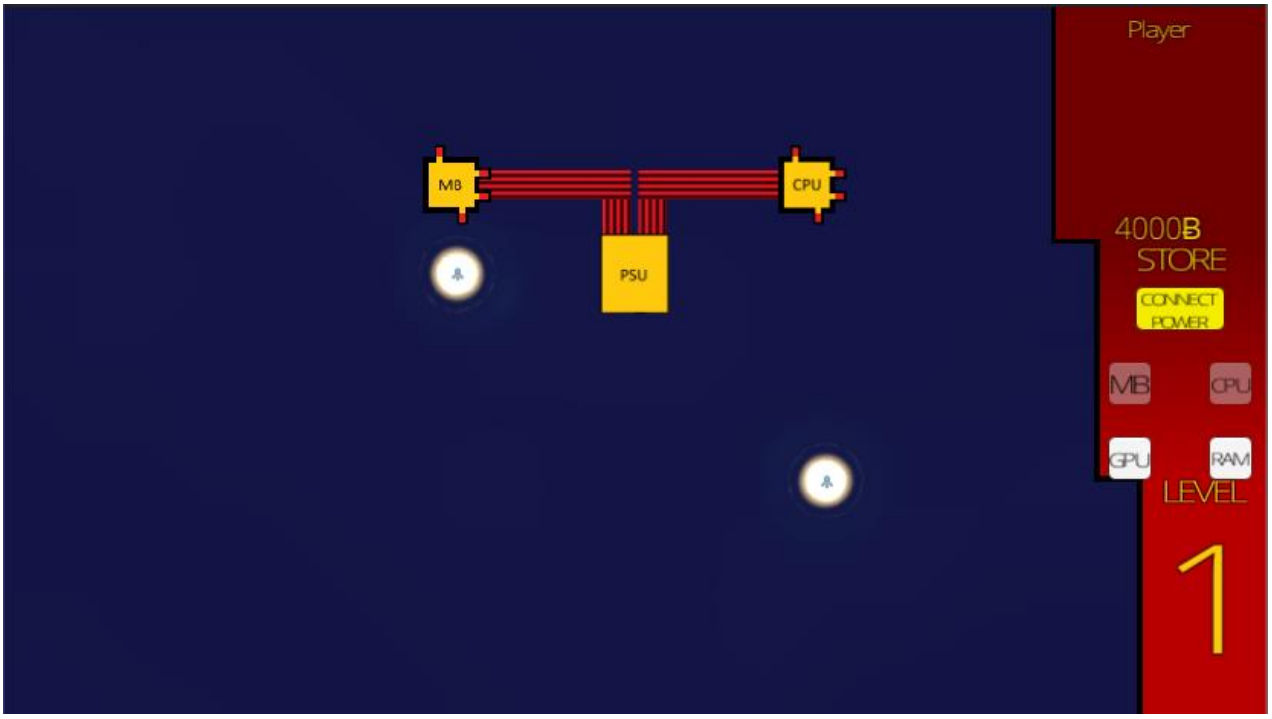
One of the major issues I encountered whilst developing the prototype was scaling issues with the games display panels. Depending on what resolution I was testing the game at, there was no way for me to accurately scale and display the panels and contents accurately. Even after numerous rounds of research into Unity UI scaling and applying the correct properties to Labels (such as setting Horizontal text wrapping on, Vertical text wrapping to overflow and setting font size to adjust to its highest possible depending on label size), Labels would still become unaligned and/or display text too large, causing it to be cut out of the display, or too small making it not stand out like hoped.



Two examples of different resolutions scaling the panel and its contents incorrectly

Eventually, I felt I was spending too much time focusing on all these adjustments to code and styling. I decided on, for prototyping purposes, to give the game a fixed Resolution of 800x600px (4:3 aspect ratio). Setting a fixed resolution ensures me that when choosing positions and sizes for panels, they will remain the same anytime I run the game.

The reason I specifically chose the 800x600 resolution, is because that is the minimum resolution you can run on any Windows 8 Operating system, meaning that any computer running that OS or above will already have the minimum resolution requirements to play this prototype.



UI being displayed correctly at 800x600 resolution

7.3 SYNCHRONIZATION WITH THE MOBILE COMPANION APP

I encountered a problem with the networking between the main game and mobile application towards the end of the development of the project. Simply put, coming up to the deadline, I felt the mobile companion app wasn't a fundamental enough component to the main area of the project. I was too short on time to implement all the Mono/.Net code required to pair the game with the companion app, as I felt that same time was better suited towards working on all the bug fixing and game balancing required to ensure the main game prototype worked as efficiently as possible.

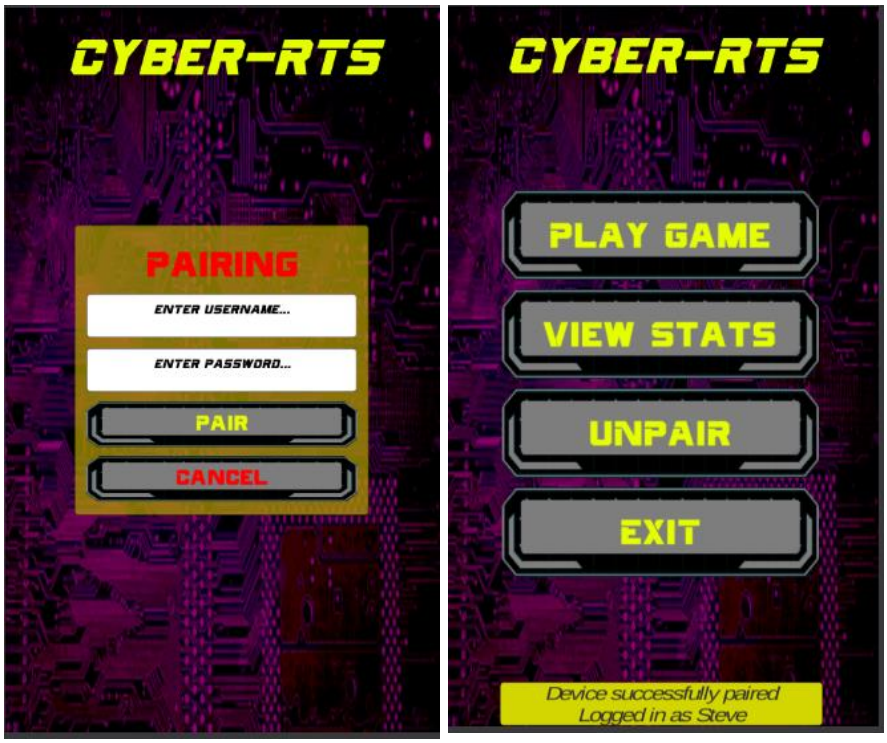
In the end, I decided to remove all the scripts and GameObject's from the game that were associated with sending and receiving information to and from the companion app as it was bringing up too many errors at build-time. I didn't want to jeopardize the ability to run the prototype because of one small non-fundamental feature that I couldn't get running.

I done the same with the mobile companion application, except that I kept the "Login" module intact as a proof of concept, operating instead as a locally-based system for storing user-data associated with the pairing mini-game.

What I mean by this is that I created a different, much smaller script that would handle a few things:

- The logging in of a previously generated player, including their total XP earned in the mini-game and high-score.
- Sending that players data to the stats panel
- The creation of a new player if a new, unique "username" is inputted.
- Parsing the logged in players' user data between the main menu scene and game scene, and then vice versa.

I also kept the stats panel in place as a proof of concept. While still showing and updating the currently logged in users total XP earned and highscore in the mini-game, the rest of the stats are just hard-coded in for demo purposes.



Logging in on the application

Displaying the stats (username, XP and highscore) for two different players. STEVE has already played the mini-game and thus has XP and a score, unlike DAWG55

8 RESULT

Upon completion of the project 400, I have resulted in two working prototypes for my "Cyber-RTS" game. The details of what features made the cut into these prototypes as well as my learning outcome from the Project 400 are listed below.

8.1 FEATURES INCLUDED

The main project deliverable for my Project 400 is the Windows component of "Cyber-RTS". This is the game itself, and while not every feature I had hoped for made it into the game, I feel I accomplished a lot with the time I had to complete it in. The features included are as such:

A good, clean User Interface with an intuitive design that wouldn't scare away first-time users. I spent a good amount of the time on the UI aspect of the game, ensuring that it is isn't too hard to follow for new-comers, while also being deep enough that experienced users won't feel it's babyish or lacking.

A deep research system and unique "base"-building mechanics. The main, two core features that help user' progress through the game. I spent a fair amount of time making sure that the base-building mechanics were balanced and that all the buildable components in the game had their own unique research trees.

A companion app which includes a simple log-in system, a proof of concept stat displaying system for each of its users and a "pair-matching" mini-game where a user tries to clear as much redundant between 2 terminals as they can within a certain time-limit.

8.2 LEARNING OUTCOMES

One of the biggest learning outcomes from the main game prototype was how to develop a User Interface that is both clean and intuitive to users. The research I done into the eight key characteristics (Clear, Concise, Familiar, Responsive, Consistent, Attractive, Efficient, Forgiving) needed to make a successful UI have given me massive insight into UI development.

Another massive learning outcome from developing this prototype is a much greater understanding of the Unity Games Development platform. To ensure I made a game that would match the standards expected for the Project 400, I done a lot of research into many of the features within Unity. What I've come away with is a much deeper understanding for how to make games in Unity. Some of the features of Unity that I have learned about from new or built knowledge upon are as follows:

- **Prefabs:** A prefab is a custom made GameObject that can contain anything from 3D models, to sprites, to scripts, to labels, to textures, to materials, to UI elements, etc. that are stored as a saved to the game and can then be instantiated anywhere in the game at any time through scripts. Prefabs were vital to the game as they allow the game to create anything from a notification to a new Computer component on the users click of a button.

- **Particle Effects:** Particle effects can range from a volcanic explosion to skid marks coming from the tires of a car. They are an important aspect for giving a game some flavor and style. I spent

some time into learning how particle effects work so I them to my game, enhancing the design. An example of particle effects used in the game are the vortex-style wave pattern coming from sources of power and bit-coin mines.

- **UI development:** Learning how to use and manipulate the many UI elements provided by Unity was essential for me to turn my game idea into a prototype success story. Here I learned how to add grid-layout panels, in-line buttons, on-click events, scaling and styling. The learning outcome from knowing how to operate Unity's UI features will be just as invaluable to me as knowing how to make clean, intuitive UI's for further pursuant dreams of becoming a successful UX engineer.

Although I never ended up including the networking component to the game, I learned a lot from the research and many trials and errors of trying to incorporate the pairing of a mobile companion app with the main game. Networking has always been my weakest aspect in programming. So thanks to all of this research and work, I feel I finally have a better understanding and some more fundamental knowledge about how I could possibly go about incorporating networking features in any future projects.

Overall, I feel this project has been extremely beneficial for me as a developer as it has provided me with a host of new and unique experiences that I never encountered in the past. The project has also opened my eyes and given me a much greater understanding of the amount of work and time that will be required of me when I decide to make further large-scale game prototypes in the future.

Screen of the final game in operation

9 REFERENCES

[1] Geryk, Bruce. "A History of Real-Time Strategy Games".
http://www.gamespot.com/gamespot/features/all/real_time

[2] Dashboards | Android Developers

https://developer.android.com/about/dashboards/index.html

[3]Unity or MonoGame for Games Development

https://www.gamedev.net/topic/673603-unity-or-monogame/

[5]Photon Networking

http://forum.unity3d.com/threads/photon-unity-networking.101734/

[6] BitRave I

https://blogs.msdn.microsoft.com/uk_faculty_connection/2014/11/26/using-unity3d-and-azure-cloud/

[7] BitRave II

http://www.deadlyfingers.net/azure/unity3d-and-cloud-backend-using-azure-mobile-services-and-prime31-plugin/

[8] Sockets and UdpClient

https://msdn.microsoft.com/en-us/library/System.Net.Sockets.UdpClient(v=vs.110).aspx

[9] ThreadNinja

https://www.assetstore.unity3d.com/en/#!/content/15717

[10] Developing a companion app for a Unity Game through Mono/.Net

http://forum.unity3d.com/threads/companion-apps-and-unity.377846/

[11] 8 Characteristics Of Successful User Interfaces

http://usabilitypost.com/2009/04/15/8-characteristics-of-successful-user-interfaces/