

Capture the Campus!
Final Report

Submitted for the BSc in
Computer Science

May 17

By
Alexander C Whitehead

Word Count: 15990

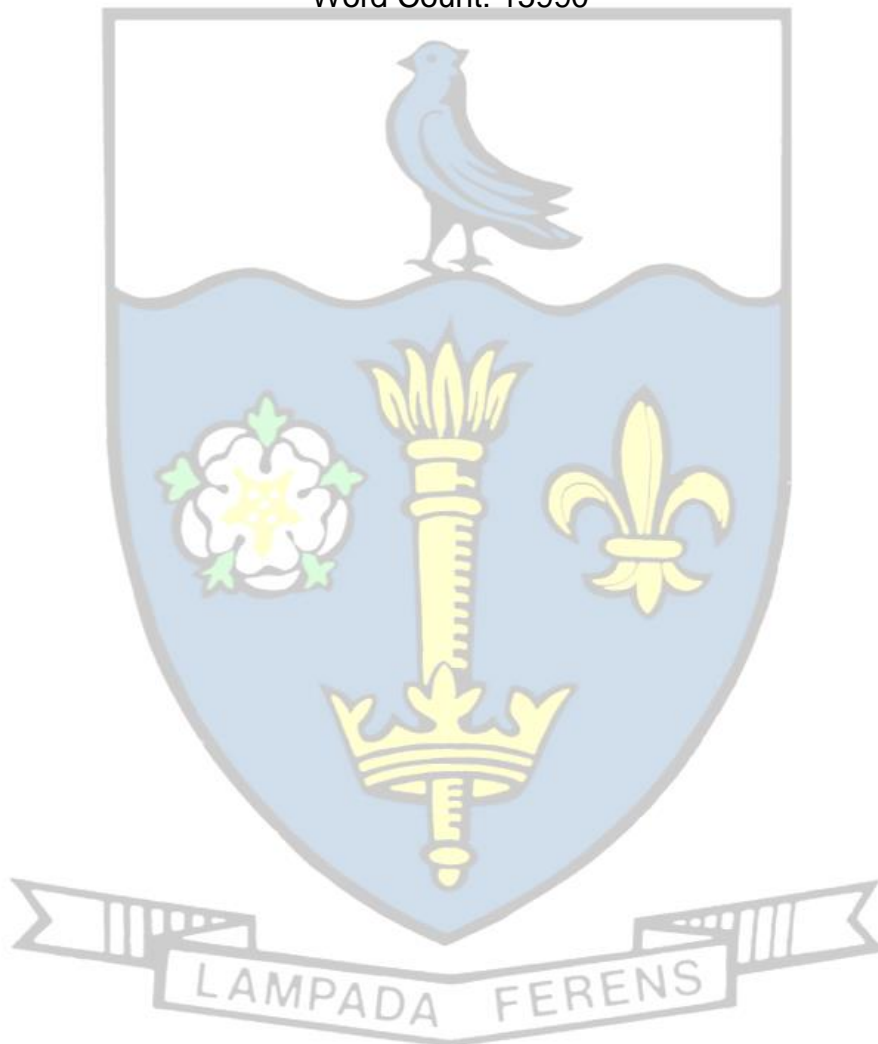


Table of Contents

1	Introduction	7
2	Aim and Objectives	9
	Objective 1 – Create a Client Library and Test Wrapper	9
	Objective 2 – Create a TCP Server and Test Wrapper	9
	Objective 3 – Create a UDP Server and Test Wrapper	10
	Objective 4 – Add Multithreading to the TCP Server	10
	Objective 5 - Add Logging to the TCP Server	10
	Objective 6 - Create a Watchdog or Heat Beat Application for the TCP Server and a Test Wrapper	10
	Objective 7 - Create a mobile application with a main menu	10
	Objective 8 - Add a Simple Map Based GPS Locator to the Mobile Application	10
	Objective 9 - Create a Method to Set the Play Area	11
	Objective 10 - Create a Method to Redefine the Play Area by the Player's Movements	11
	Objective 11 - Add Scoring and Game Ending Logic to the Game	11
	Objective 12 - Add a Game Over Screen	11
	Objective 13 - Add a Representation for the Enemy to the Game	11
	Objective 14 - Create a Method That Can Randomly Generate the Enemy a New Location	11
	Objective 15 - Create a Method That Allows the Enemy to Randomly Wander Within the Play Area	12
	Objective 16 - Create a Method That Deals with an Intersection between the Enemy and the Player	12
	Objective 17 - Add a Lobby for Local Multiplayer	12
	Objective 18 - Create a Method of Connection between Host and Players	12
	Objective 19 - Add Logic to Allow Local Multiplayer	12
3	Background	13
3.1	Problem Context	13
3.1.1	Alternative Solutions	13
3.1.2	Tracking	16
3.2	Comparison of Technologies	18
3.2.1	Platform	18
3.2.2	Language	20
3.2.3	Networking	22
3.2.3.1	Architecture	22
3.2.3.2	Protocol	24
4	Technical Development	26
4.1	System Design	26

4.1.1	Networking Design	29
4.1.1.1	Client Design	30
4.1.1.2	Watchdog Design	33
4.1.1.3	TCPServer Design	35
4.1.1.4	UDPServer Design.....	37
4.1.2	CaptureTheCampus Design	39
4.1.2.1	Game Design.....	39
4.1.2.2	Menu Design.....	42
4.1.2.3	Score Design	43
4.1.2.4	Search Design	44
4.1.2.5	Set Design	46
4.1.2.6	Static Design.....	47
4.2	User Interface Design	49
4.3	System Implementation	62
4.3.1	Processes and Methodology	62
4.3.2	Version Control.....	64
4.3.3	Polygon Clipping Algorithm.....	65
4.3.4	Polygon Tessellation Algorithm	67
4.3.5	Wandering Algorithm.....	68
4.4	Test Design.....	70
5	Evaluation	75
5.1	Project Achievements	75
5.1.1	Final Task List	76
5.1.2	Final Time Plan	81
5.1.3	Test Evaluation.....	86
5.2	Further Work.....	90
6	Conclusion.....	91
Appendix A:	Capture the Campus!	92
Appendix B:	Code Map Legend.....	93
Appendix C:	Code Map	94
Appendix D:	Networking Code Map.....	95
Appendix E:	CaptureTheCampus Code Map.....	96
Appendix F:	Polygon Clipping Code.....	97
Appendix G:	Polygon Tessellation Code.....	105
Appendix H:	Wandering Code	109
Appendix I:	Initial Task List	114
Appendix J:	Initial Time Plan.....	116
Appendix K:	Interim Task List.....	118

Appendix L: Interim Time Plan	121
Appendix M: Risk Analysis.....	123
Acknowledgements	126
References	127

Table of Figures

Figure 1: This is an image showing the gameplay of Qix (The International Arcade Museum, Museum of the Game, 2016).....	8
Figure 2: This image shows a standard game screen for Pokémon GO (Wikipedia, 2017).	14
Figure 4: This image shows a standard game screen for Ingress (Wikipedia, 2017).	15
Figure 3: This image shows a standard game screen for Ingress (Google, 2017)	15
Figure 5: This image shows the orbits of some GPS satellites	16
Figure 6: This image shows a visualisation of the act of GPS trilateration (openclipart, 2014)	17
Figure 7: This image shows the current market share of mobile operating systems (Statista, 2016).	18
Figure 8: This image shows an example of a standard Android Studio integrated development environment window (Wikipedia, 2017).....	20
Figure 9: This image shows an example of a standard Visual Studio integrated development environment window (Wikipedia, 2017).....	20
Figure 10: This image shows an example of a standard Xcode integrated development environment window (Apple, 2017).	21
Figure 11: This image shows a visual model of the difference between a client/server model on the left and a peer-to-peer model on the right (Philips, 2014).	22
Figure 12: This image shows the latency involved in sending a message using the different transport layer protocols (Taing, 2011).....	24
Figure 13: This image shows the throughput in Mbps of the different transport layer protocols (Taing, 2011).	25
Figure 14: This image shows a graphical representation of the workflow and a stepwise diagram of activities and actions of the system.	26
Figure 15: This image shows a diagram that describes the structure of the Client subsystem by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Wikipedia, 2016).	30
Figure 16: This image shows a diagram that describes the structure of the Watchdog subsystem by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Wikipedia, 2016).	33
Figure 17: This image shows a diagram that describes the structure of the TCPServer subsystem by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Wikipedia, 2016).	35
Figure 18: This image shows a diagram that describes the structure of the UDPServer subsystem by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Wikipedia, 2016).	37
Figure 19: This image shows a diagram that describes the structure of the Game subsystem by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Wikipedia, 2016).	39
Figure 20: This image shows a diagram that describes the structure of the Menu subsystem by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Wikipedia, 2016).	42
Figure 21: This image shows a diagram that describes the structure of the Score subsystem by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Wikipedia, 2016).	43
Figure 22: This image shows a diagram that describes the structure of the Search subsystem by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Wikipedia, 2016).	44

Figure 23: This image shows a diagram that describes the structure of the Set subsystem by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Wikipedia, 2016).	46
Figure 24: This image shows a diagram that describes the structure of the Static subsystem by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Wikipedia, 2016).	47
Figure 25: This image shows a representation of the simplest interactions a user can have with the system.	49
Figure 26: This image shows two examples of menu styles used in mobile application (Vogel, 2016)	50
Figure 27: This image shows examples of the user interface for the Android launcher Arrow Launcher by Microsoft (Funk, 2015).....	51
Figure 28: This image shows an example of a retro arcade style menu (Park Productions, 2015)	51
Figure 29: This image shows a standard game screen for the Google Maps API based game War2Map (Dempsey, 2013)	52
Figure 30: This image shows the user interface for the Menu activity.....	53
Figure 31: This image shows the user interface for the Set activity before any markers or flags have been set.....	54
Figure 32: This image shows the user interface for the Set activity after the markers or flags have been set.	55
Figure 33: This image shows the user interface for the Game activity just after the game has been started.....	56
Figure 34: This image shows the user interface for the Game activity as a player is attempting to capture an area.	57
Figure 35: This image shows the user interface for the Game activity just after a player has captured an area.....	58
Figure 36: This image shows the user interface for the Score activity.	59
Figure 37: This image shows the user interface for the Host activity.	60
Figure 38: This image shows the user interface for the Join activity.....	61
Figure 39: This image shows a collage of the three main software development models (Wikipedia, 2017).	62
Figure 40: This image shows a graphical representation of the iterative and incremental development software development model (Wikipedia, 2017).	63
Figure 41: This image shows a graphical representation of the commit history for this project.	64
Figure 42: This image shows a diagram that describes the structure of the Tests subsystem by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Wikipedia, 2016).	70
Figure 43: This image shows the legend or key for the code map diagrams.	93
Figure 44: This image shows a diagram that describes the structure of the system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Wikipedia, 2016).	94
Figure 45: This image shows a diagram that describes the structure of the Networking subsystem by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Wikipedia, 2016).	95
Figure 46: This image shows a diagram that describes the structure of the CaptureTheCampus subsystem by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Wikipedia, 2016).....	96

1 Introduction

This is a report describing the final research and development stages of the project to build Capture the Campus!.

This report includes:

1. A section introducing the project and covering its main aims and objectives.
2. A section discussing background research conducted. Including a subsection describing the context of the project, a subsection introducing alternative solutions to similar projects, a subsection comparing relevant development technologies and a subsection comparing known solutions to anticipated problems.
3. A section discussing the design and development of the project. Including a subsection describing the design of each part of Capture the Campus!, a subsection describing the design of the user interface, a subsection discussing any tests implemented and a subsection discussing the implementation of the solutions discussed in section 2.
4. A section evaluating the project's success in hitting the aims outlined in section 1 and a subsection describing any further work that could be performed to complete unachieved goals.
5. A section summarising the project.

Capture the Campus! is a GPS based mobile game influenced by the classic 80's arcade game Qix (System 16, 2014) and to a lesser extent Pokémon GO (Niantic, 2016), and Ingress (Niantic, 2016).

In the game of Qix the objective is to steer a player character around the periphery of a square play area before then traveling across this play area all while avoiding multiple computer-controlled enemies.

Once the player character has crossed the play area the play area is then redefined as the larger of the two polygons that are created by splitting the play area into two along the path travelled by the player character. A score is awarded to the player character based on the size of the polygon removed from the play area.

The enemies mentioned above wander randomly around within the play area, if an enemy intersects the player character's path the player character's path is reset and they lose a life.

An example of a standard Qix game screen can be seen below, including showing multiple scoring polygons that have been split by the player character away from the main play area (Figure 1).



Figure 1: This is an image showing the gameplay of Qix (The International Arcade Museum, Museum of the Game, 2016).

The objective of Capture the Campus! is like that of Qix, as described above. However, rather than controlling a player character with something like a mouse and keyboard or a game controller the player character is controlled by the player's physical movement in the real world.

The play area is defined using latitude, longitude coordinates, and to capture parts of this play area the player must move physically from one side of it to the other.

The game is run from a mobile device in the player's possession and their location is tracked using GPS data from the device's GPS sensor.

As in Qix there are computer-controlled enemies, these enemies can kill the player but rather than removing a life, points are deducted from their score.

The game ends after a predetermined area has been captured.

The original specification for Capture the Campus! can be seen below (Appendix A:).

2 Aim and Objectives

The aim of this project is:

To create a mobile platform based augmented reality, Qix like, area control game

This aim was achieved by completing the following objectives:

1. Create a Client Library and Test Wrapper
2. Create a TCP Server and Test Wrapper
3. Create a UDP Server and Test Wrapper
4. Add Multithreading to the TCP Server
5. Add Logging to the TCP Server
6. Create a Watchdog or Heat Beat Application for the TCP Server and a Test Wrapper
7. Create a mobile application with a main menu
8. Add a Simple Map Based GPS Locator to the Mobile Application
9. Create a Method to Set the Play Area
10. Create a Method to Redefine the Play Area by the Player's Movements
11. Add Scoring and Game Ending Logic to the Game
12. Add a Game Over Screen.
13. Add a Representation for the Enemy to the Game
14. Create a Method That Can Randomly Generate the Enemy a New Location
15. Create a Method That Allows the Enemy to Randomly Wander Within the Play Area
16. Create a Method That Deals with an Intersection between the Enemy and the Player
17. Add a Lobby for Local Multiplayer
18. Create a Method of Connection between Host and Players
19. Add Logic to Allow Local Multiplayer

Objective 1 – Create a Client Library and Test Wrapper

Before a server could be created, a basic client library was first developed to test the TCP and UDP servers with.

The client was built first as examples of working TCP and UDP servers were easy to come by online to test the client's functionality against.

The client can send messages using TCP and UDP protocols. The client can send TCP messages using any IP address or port.

Development of the client did not end until both it and the servers were completed.

The client library was integrated into the game once completed, additionally it can be reused in future projects.

A wrapper for the client was created to allow it to be used as both a library and as an application for integration and testing.

Objective 2 – Create a TCP Server and Test Wrapper

The TCP server stores the vertices of the initial play area, the position of the enemy and the current location of each player.

This information is used to synchronize all instances of the current local multiplayer game.

The TCP server can be run from a phone to allow multiple concurrent instances of multiplayer games.

The TCP server can be opened on any port.

A wrapper for the TCP server was created to allow it to be used as both a library and as an application for integration and testing.

This will be discussed further in the Networking (3.2.3) section of the Background.

Objective 3 – Create a UDP Server and Test Wrapper

The UDP server is used as an initial contact point for the client, the UDP server can broadcast a coded message containing the IP address of the TCP server. The coded message should help to eliminate malicious attacks on the TCP server.

The UDP server can run from a phone to allow multiple concurrent instances of multiplayer games.

The UDP server can be opened on any port.

A wrapper for the UDP server was created to allow it to be used as both a library and as an application for integration and testing.

This will be discussed further in the Networking (3.2.3) section of the Background below.

Objective 4 – Add Multithreading to the TCP Server

The TCP server has had multithreading capabilities added; the TCP server can accept multiple TCP requests simultaneously to support several players concurrently.

Objective 5 - Add Logging to the TCP Server

The TCP server should be able to write a log of all messages for debugging purposes.

The log should be able to be created at any file directory.

The TCP server should be able to rebuild itself after a fatal error, for instance a crash, from this log file.

Objective 6 - Create a Watchdog or Heart Beat Application for the TCP Server and a Test Wrapper

A Watchdog or Heart Beat application was created; this application checks to see if the game has crashed and cleans up, if it has, by stopping the servers.

A wrapper for the Watchdog or Heart Beat application was created to allow it to be used as both a library and as an application for integration and testing.

Objective 7 - Create a mobile application with a main menu

An initial application was created to build the games functionality upon.

Before the menu is launched, all dependencies for the game are checked and the application does not run if a dependency is found to be missing.

The main menu has the capability to start a single player game, start a local multiplayer game as a host or join an already existing local multiplayer instance.

Objective 8 - Add a Simple Map Based GPS Locator to the Mobile Application

Before the game logic was added to the application a simple map based GPS locator application was developed, this displays the device's location on a map. The map translates with the device's movement.

This application was then transformed by the addition of gameplay into the intended product.

Objective 9 - Create a Method to Set the Play Area

A method was created which is used before a game instance is initialised to set the initial play area for the game.

To be in keeping with the influence of Qix the initial play area is limited to being a square or rectangle.

Objective 10 - Create a Method to Redefine the Play Area by the Player's Movements

A method has been created which is used to track a player's path through the play area. Once the player exits the play area it redefines the play area as the larger of the two polygons created by splitting the play area into two along the path travelled by the player. If a player crosses their own path the method removes the loop from the player's path as otherwise more than two polygons would be created when splitting the play area. The area removed from the play area is filled with the player character's colour.

Objective 11 - Add Scoring and Game Ending Logic to the Game

When a player removes a polygon from the play area, their score is increased by an amount relative to the polygon's size.

The game ends once the play area's size has been reduced by a certain amount.

Objective 12 - Add a Game Over Screen

A screen was created that is displayed to the player once a game has ended.

The game over screen displays the player's score and allows them to return to the main menu to play again.

Objective 13 - Add a Representation for the Enemy to the Game

Something that is used to represent the computer-controlled enemy was added to the play area.

If the enemy finds itself outside of the play area, for instance, if the enemy is located within a polygon removed from the play area then the enemy moves itself to be within the play area again.

Objective 14 - Create a Method That Can Randomly Generate the Enemy a New Location

A method was created which can find a random location within any given play area, this is because no point within the initial play area can be assumed to always exist within the redefined play area

This is used as an initial location for the enemy or to relocate the enemy if it finds itself outside of the play area.

Objective 15 - Create a Method That Allows the Enemy to Randomly Wander Within the Play Area

To be in keeping with the influence of Qix the computer-controlled enemy follows a wandering path like the eponymous Qix enemy from Qix.

This path follows a cone of possible moves relative to the normal vector of the previous move of the enemy, this forces the enemy to move around the play area rather than, as would be possible otherwise, vibrating back and forth eternally.

Objective 16 - Create a Method That Deals with an Intersection between the Enemy and the Player

A method was added that removes the player's path and deducts an amount from their score if the enemy intersects any point along their path.

A lives system was not implemented as lives systems were originally developed to encourage people to pay to play arcade games more frequently (Spikejumper2, 2017).

A lives system would also clash with the ethos of a multiplayer game as some players could end up without any more lives well before their peers, thus rendering them out of the game.

Objective 17 - Add a Lobby for Local Multiplayer

A lobby was created where players wait for their local multiplayer game to start, while the servers on the backend connect and synchronise their devices.

In case the UDP server fails in establishing an initial connection the option to input the IP address of the target host is allowed, thus the IP address of the host is also displayed on the host device.

Objective 18 - Create a Method of Connection between Host and Players

A backend method was created which while the players wait in the lobby synchronises their devices with the vertices of the initial play area, the position of the enemy and the current location of each player.

This method also responds to the Watchdog or Heart Beat application to stop it from prematurely closing the TCP server.

Objective 19 - Add Logic to Allow Local Multiplayer

The gameplay logic was adapted to allow and track multiple simultaneous players.

Each instance of the local multiplayer game is also synchronised with the vertices of the initial play area, the position of the enemy and the current location of each player.

The game also responds to the Watchdog or Heart Beat application to stop it from prematurely closing the TCP server.

3 Background

3.1 Problem Context

3.1.1 Alternative Solutions

Pokémon GO is a game developed by Niantic. In Pokémon GO, the objective is to physically travel around the real world attempting to find and capture objects (Niantic, 2016).

Pokémon GO is an application developed for mobile devices, it operates using a mobile device's GPS sensor to find the player's location and synchronises all the player's devices in the world using a centralised server (Niantic, 2016) (Lawson, 2016).

In Pokémon GO when an objective is captured its data is added to a database controlled by the player but it is not removed from the map and other players are free to capture it.

The user interface for Pokémon GO represents a player's position using a human avatar that faces in the direction of travel and moves with a running motion from a player's previous position to their current position when they move. Objectives are marked on the Pokémon GO map using a marker and can be interacted with when a player comes within a set distance or radius of them.

The gameplay of Pokémon GO borrows voraciously from capture the flag style games and could even be considered a semi pseudo geocaching game (Ordnance Survey, 2017) (Niantic, 2016).

An example of a standard Pokémon GO game screen can be seen below, including showing a player's avatar and multiple objective markers (Figure 2).



Figure 2: This image shows a standard game screen for Pokémon GO (Wikipedia, 2017).

Ingress is also a game developed by Niantic. In Ingress, the objective is to physically travel around the real world attempting to capture objects and control areas.

Ingress is an application developed for mobile devices, it operates using a mobile device's GPS sensor to find the player's location and synchronises the player's devices in the world using a centralised server (Niantic, 2016).

In Ingress, the objective relies on controlling large swathes of the map by capturing nodes on the map. Players directly compete to hold the largest area; this contrasts with Pokémon GO where players are not in direct competition with each other (Whelan, 2014).

The user interface for Ingress represents a player's position using a marker that points in the direction of travel. Objectives are marked on the Ingress map using a glowing marker and can be interacted with when a player comes within a set distance or radius of them.

An example of a standard Ingress game screen can be seen below, including showing the areas controlled by two teams in one area of the map (Figure 3) (Figure 4).

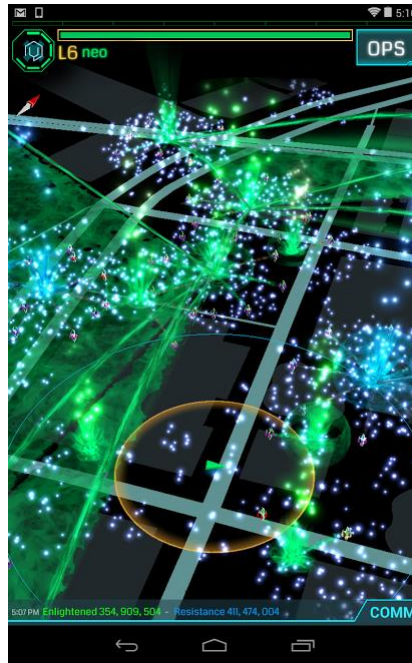


Figure 4: This image shows a standard game screen for Ingress (Google, 2017)

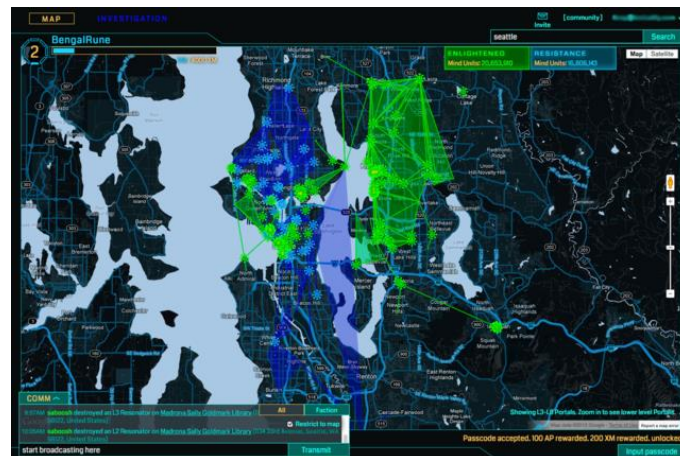


Figure 3: This image shows a standard game screen for Ingress (Wikipedia, 2017).

One useful critique of Pokémon GO and Ingress would be that even though the appeal of this game is as an augmented reality experience, there should be some none augmented reality or single player content to aid in accessibility for those that are not mobile (Credits, Extra, 2016). However, this kind of gameplay is beyond the scope of this project. Another would be that Pokémon GO is a reskinning or asset swap of Ingress without any improvements if not actually a step backwards in technical advancements (Madhavan, 2016).

Ideas and themes that were adapted from Pokémon GO and Ingress include; the server and database methods used in Pokémon GO and Ingress to synchronise captured objects between players of the game and the area capturing mechanics and user interface used in Ingress.

3.1.2 Tracking

Imperative to the implementation of this project was for the mobile device, on which Capture the Campus! runs, to be aware of its own location. In this case, the Global Positioning System (Brown & Sturza, 1995) was used; this is because a large percentage of modern mobile devices contain some form of Global Positioning System sensor (Zhao, 2002).

The Global Positioning System is usually accurate regardless of weather conditions (Ian A.R. Hulbert, 2001). The Global Positioning System will work wherever there is an unobstructed line of sight to at least four Global Positioning System satellites.

Unfortunately, access to at least four satellites is required as each satellite is needed to pinpoint the user in one dimension of space or time (Crato, 2010).

The Global Positioning System can operate with only line of sight rather than through a mobile or internet connection because it works via a direct connection to the Global Positioning System satellites, which each contain their own atomic clock (Kaplan & Hegarty, 2006).

The position of the Global Positioning System satellites in orbit around the earth can be seen in the image below. (Figure 5)

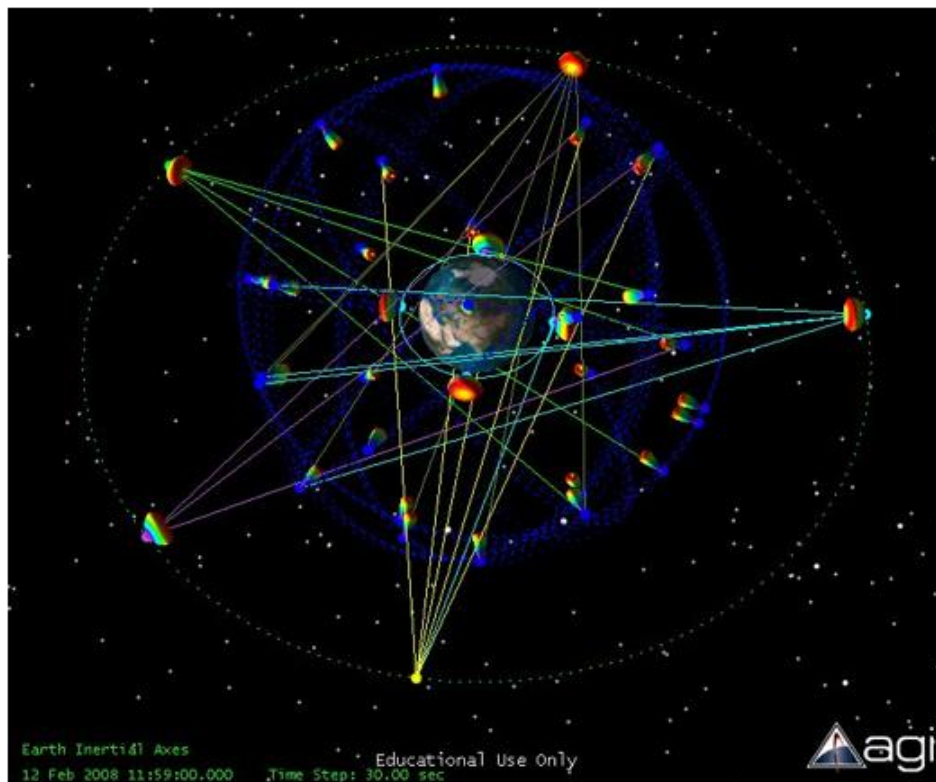


Figure 5: This image shows the orbits of some GPS satellites

The Global Positioning System works using modulated electromagnetic signals. These signals are first sent from the sensor in the host device to the Global Positioning System satellites orbiting the earth; this initial signal carries a pseudorandom code that verifies the identity of the sender.

After an initial connection is established an additional signal is sent which requests the time as stated by the atomic clocks contained within the orbiting Global Positioning System

satellites, these satellites then respond with a similarly modulated signal containing the exact time that they believe it to be plus their position in orbit around the earth (Mooney, 1985). The sender, after having waited to receive a response from a suitable number of Global Positioning System satellites in range, then uses the time as stated in each response to calculate the flight time of each message.

Knowing the flight time, the speed of the communication and the location in three-dimensional space of the Global Positioning System satellite that sent the response, the device can use basic trigonometry to determine a spherical area around each Global Positioning System satellite in which the device may be located. The point where the spheres of all the Global Positioning System satellites intersect is the point where the device is located (Ackermann, 1994).

This method is known as trilateration and a graphical representation of the Global Positioning System satellites and their possible location spheres can be seen in the image below (Tarasenko, 2009) (Figure 6).

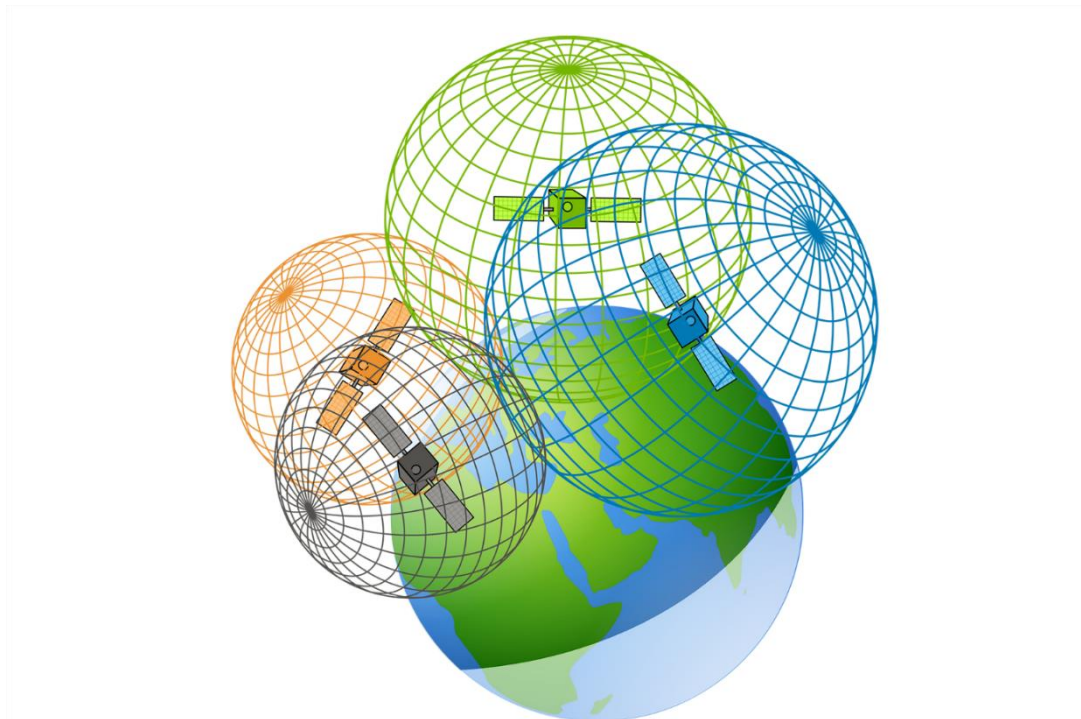


Figure 6: This image shows a visualisation of the act of GPS trilateration (openclipart, 2014)

Using modern programming languages and APIs, like C# or Java, it is a rather trivial affair to implement accurate Global Positioning System tracking (Anna, 2016) (Xamarin, 2017).

For instance, the Google Play Services SDK available for Android development allows the use of the devices Global Positioning System sensor through the Google Location Services API and the Fused Location Provider API.

The Google Location Services API provides access to the phones Global Positioning System sensor, which is highly accurate but drains the devices battery and will not work without line of sight (Google, 2017).

The Fused Location Provider API augments the Google Location Services API with device location information attained using the devices mobile or internet connections, if applicable. This allows the device to attain highly accurate location information without draining the mobile devices battery and even when there is not direct line of sight to the Global Positioning System satellites (Google, 2016).

3.2 Comparison of Technologies

3.2.1 Platform

Before development began with aplomb, a target platform needed to be chosen for the application to be developed for. One of the requirements for this project was that the target platform needed to be a mobile platform. Thus, a desktop or laptop computer platform was almost immediately ruled out, as it would be either impossible or laborious to physically transport something with such size, weight and power requirements around to play. In addition, these kinds of platforms also do not tend to contain the required GPS sensors.

Therefore, the choice of target platform was reduced even further with these requirements in mind to only include; Android phones, iOS phones, Windows phones, and GPS handhelds.

From these choices, the GPS handhelds were safely disregarded as it was assumed that the project should be easily distributed. For the project to be run on a GPS handheld a specific handheld would have had to be selected and the project would have needed to be developed using the manufacturers APIs and SDKs to run solely on that device which would have not met the requirements for the specification (Garmin, 2008).

This left the choice of target platform as a tossup between the three-main mobile phone operating systems; Android, iOS, and Windows phone. Unfortunately, because Windows phone has such a low market share it was also eliminated as documentation and support would have been lacking thus increasing the difficulty and length of development and drastically impacting the distribution of the project (Statista, 2016). A graph showing the market share of the main mobile operating systems can be seen below (Figure 7).

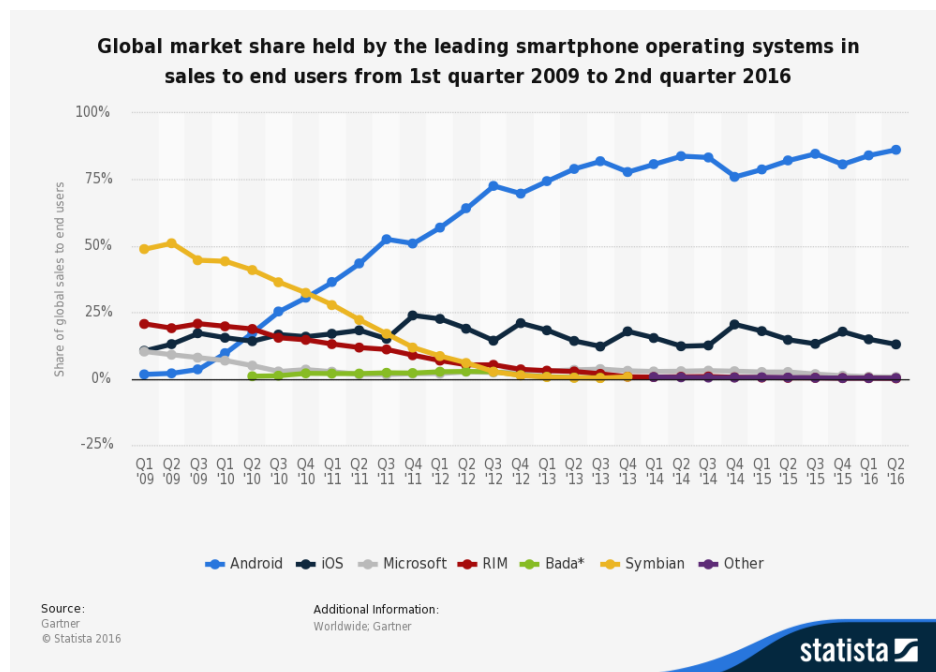


Figure 7: This image shows the current market share of mobile operating systems (Statista, 2016).

After the elimination of Windows phone, only Android and iOS remained as contenders for the target platform for the project.

On the surface, it may appear that Android would have been the obvious choice over iOS as it has a far greater market share. However, it had been noted in numerous reports that on average iOS users tend to spend substantially more money on quality content more than making up for their poor market share (Stenovec, 2015) (Sinicki, 2016).

Therefore, it mattered more what the exact goal of this project was regarding distribution and profitability. If the goal of this project were mass distribution, then Android would have been far the better choice. However, if the goal were to be profitable, then iOS would have been a better target platform (Sunny, 2016).

Rather than attempting to separate the candidates on business ventures alone it would have been more fruitful to acknowledge the experience of having to developing in and for their respective environments.

Designing the user interface for an Android application is far easier than designing the user interface for an iOS application as the design specification for Android applications are much more thoroughly documented than the design specifications for iOS applications (O'Sullivan, 2015).

The process to publish applications to the Google Play Store and to receive approval from the Google Play store is far simpler and takes less time on average than it does to publish applications to the Apple App Store. This is because anyone can upload their Android application to the Google Play store and receive approval in minutes by an autonomous system put into place by Google, whereas Apple insists on having each application tested by a real-life human employee before issuing approval for the application to enter the Apple App Store. This process takes a very long time and has a high rejection rate (Sinicki, 2016). Obviously both platforms have their merits, given the features of both operating systems it was too close to call a superior platform.

3.2.2 Language

For development to finally begin on the project a programming language needed to be chosen for the project to be written in.

As discussed in the Platform section (3.2.1) previously the platform to develop the project for had been narrowed down to either Android or iOS.

If the decision was made to develop for Android the languages available to write the project in would have been Java using the Android Studio integrated development environment or in C# via the Xamarin for Visual Studio integrated development environment.

Examples of the environment used to write in these languages can be seen in the images below (Figure 8) (Figure 9).

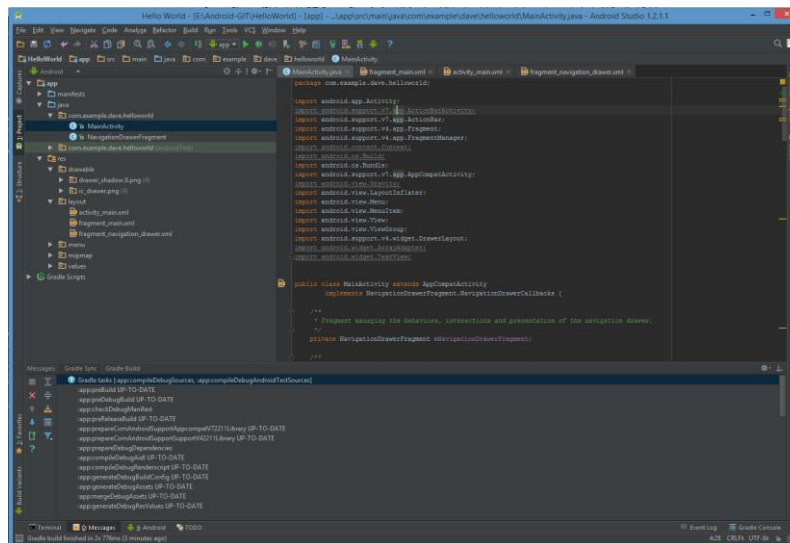


Figure 8: This image shows an example of a standard Android Studio integrated development environment window (Wikipedia, 2017).

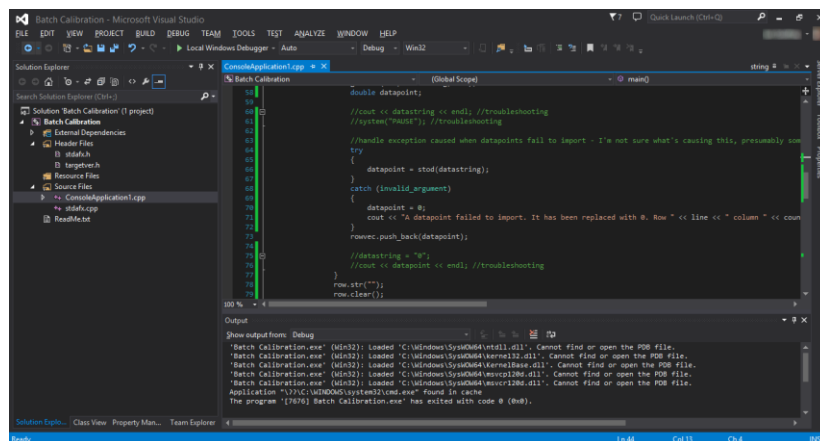


Figure 9: This image shows an example of a standard Visual Studio integrated development environment window (Wikipedia, 2017).

The Android compilers for both the Java and Xamarin programming languages produce native code solutions for Android devices, these native code solutions run directly on the target devices processor without an emulation layer (Google, n.d.) (Xamarin Inc., 2016).

An emulation layer would have slowed down the execution of the application as it sits between the compiled code and the processor translating the compiled code in real time into something the processor can understand (Rouse, 2006).

Because the Java and Xamarin programming languages both produce native Android solutions, they should both produce applications which run at similar speeds. However, Xamarin also allows for the use of the standard Microsoft .NET libraries, which Java does not (Montemagno, 2016). Thus, due to the project's developer already being familiar with the standard Microsoft .NET libraries a solution would have been able to be created in Xamarin at a greater speed and to a higher quality standard than in Java because of the lack of a bottleneck associated with the project's developer having to learn a new programming language (Scholtz, 1992).

However, if the decision was to develop for iOS instead, the languages available to write the project in were Swift using the Xcode integrated development environment or in C# via the Xamarin for Visual Studio integrated development environment.

Examples of the environment used to write in these languages can be seen in the following images (Figure 10) (Figure 9).

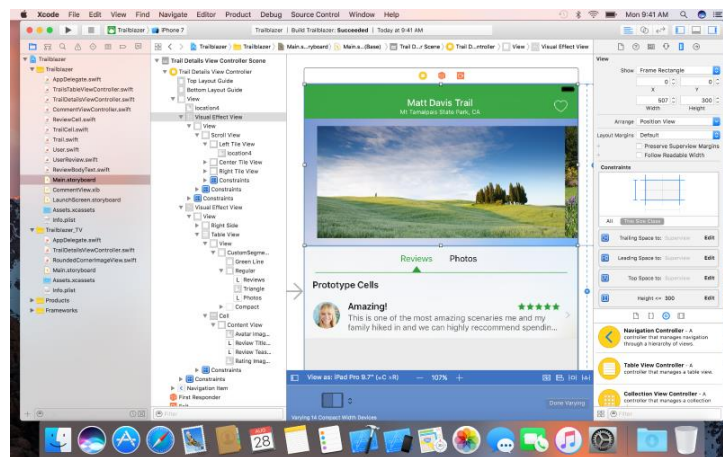


Figure 10: This image shows an example of a standard Xcode integrated development environment window (Apple, 2017).

The iOS compilers for both the Swift and Xamarin programming languages produce native code solutions for iOS devices (Apple, 2016) (Xamarin Inc., 2016).

Because the Swift and Xamarin programming languages both produce native iOS solutions, they would have both produce applications which ran at similar speeds. However, Xamarin would also allow for the use of the standard Microsoft .NET libraries, which Swift does not (Montemagno, 2016). Thus, due to the project's developer already being familiar with the standard Microsoft .NET libraries a solution would have been able to be created in Xamarin at a greater speed and to a higher quality standard than in Swift.

Another advantageous feature of Xamarin is that it can be used to write programs for both Android and iOS using a shared common code base, because of this it was unnecessary to choose between both Android and iOS as the target platform.

This meant that the theoretical user base for the project rose from 86.2% or 12.9% alone targeting either Android or iOS respectively to a combined total user base of 99.1% (Xamarin Inc., 2016) (Statista, 2016).

Therefore, C# via the Xamarin for Visual Studio integrated development environment was safely chosen as the programming language of choice to write this project in.

3.2.3 Networking

3.2.3.1 Architecture

For the implementation of a multiplayer version of the project some form of networking component was required to synchronise all current players.

The networking component, as discussed in the Language section (3.2.2), was written in the Xamarin programming language as it allowed for the development of a high quality, cross platform solution (Xamarin Inc., 2016).

Xamarin would also allow the network component to be run from either a desktop or a mobile device using a wrapper program that allows for easy debugging and rapid prototyping (Xamarin Inc., 2016).

The ability for the networking component to be able to be run from either a desktop or a mobile device was desirable as it allowed the option for the project's developer to define either a predefined, offsite, central server for all users to use or multiple local servers that could be run directly from the player's mobile device.

There were several models and schools of thought on how networking components could be implemented and the way they go about synchronising data between themselves.

One network model would be peer-to-peer architecture. In a peer-to-peer network each device on a network synchronises directly with every other device on the network equally; this form of network structure generally has no central storage or user authentication (Posey, 2000).

Another network model would be client/server architecture where there are separate dedicated clients and servers; this form of network structure works solely through a central storage server and usually has some form of user authentication (Posey, 2000).

These two different architectures can be visualised in the image below (Figure 11).

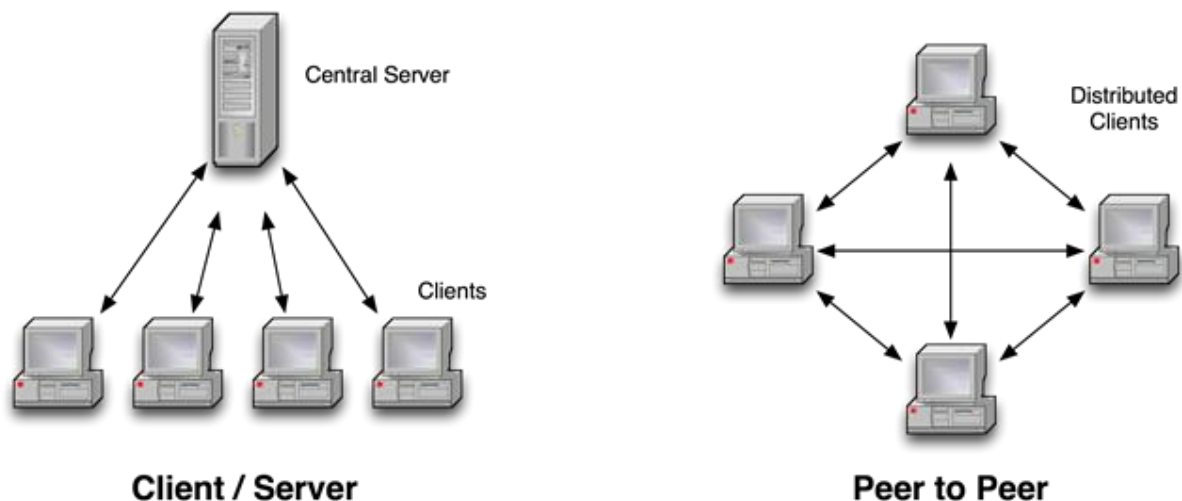


Figure 11: This image shows a visual model of the difference between a client/server model on the left and a peer-to-peer model on the right (Philips, 2014).

Peer-to-peer networks operate at a reasonable speed with few clients over short distance, similar speed network connections (Baccelli, et al., 2013).

However, peer-to-peer networks struggle when scaled up to a larger number of clients, this is due to the whole network being bottlenecked by the speed of the slowest network connection in the network (Baccelli, et al., 2013).

Conversely, client/server networks can handle many users simultaneously with ease. This is because each client in a client/server network has their own private connection to the server; the performance of each client's connection is only affected by their own connection to the server (Sparrow, 2017).

However, client/server networks usually require an offsite server or servers be dedicated to the application always to operate, if these servers become unavailable the whole network is compromised (Sparrow, 2017).

For this project, a client/server model was most effective, this was because the player's mobile device on which the application would be running would have an unreliable network connection (Zhang & Soong, 2006). Unreliable mobile network connections are partly due to frequent network handovers where a device moves between the boundaries of areas covered by different networks (Ali, et al., n.d.).

Therefore, if a peer-to-peer architecture was selected the poor mobile network connection between the players would cause the application to constantly halt while waiting for a response from a player who is no longer in fact connected to the network.

In addition, because the application was designed for mobile devices it could be assumed that the players would be on a rated connection, where every piece of data could cost the players money to send, it would have been unnecessary to cause the players avoidable cost.

Thus again, a client/server model would be the most effective implementation as it requires the players to send and receive only one message to synchronise data across all devices, this was because all players would communicate with one central server where all data would be stored (Bernstein & Goodman, 1981).

3.2.3.2 Protocol

To create a network connection between client and server a transport layer protocol was required to carry each message. There were several protocols that the client and server could have been written to send and receive, each with its own uses, advantages and disadvantages.

The protocols that the client and server could have been written to send and receive include: The Remote Method Invocation protocol, a high latency, low reliability, low throughput protocol (Taing, 2011).

The User Datagram Protocol, a low latency, lower reliability, high throughput protocol with the ability to broadcast a message to all IP addresses on the local network (TechTarget, 2015) (Seguin, 2014).

The Transmission Control Protocol, a higher latency, high reliability connection-oriented protocol that if there were a network connection between two devices any message sent using it would be guaranteed to arrive (TechTarget, 2014) (Diffen, n.d.).

The relative performance of these three transport layer protocols can be seen in the two diagrams below (Figure 12) (Figure 13).

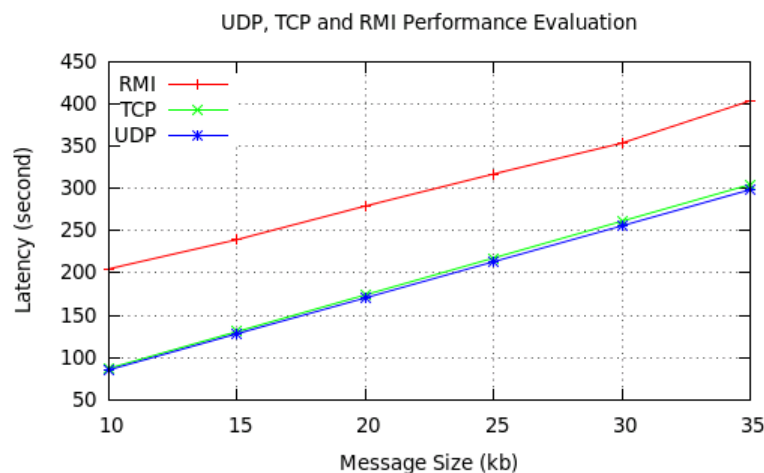


Figure 12: This image shows the latency involved in sending a message using the different transport layer protocols (Taing, 2011).

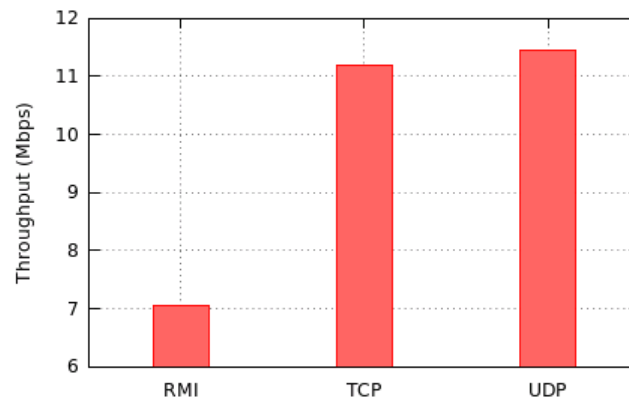


Figure 13: This image shows the throughput in Mbps of the different transport layer protocols (Taing, 2011).

This project required a method by which to establish an initial connection between the clients and server, then once a connection had been established this project required a method by which to synchronise data between the clients and server.

The User Datagram Protocol was perfect for establishing an initial connection between the clients and server because of its low latency and ability to broadcast a message across all IP addresses on the local network (Mey, n.d.).

The Transmission Control Protocol was then useful once the initial connection had been established to synchronise data between the clients and server, this was because the Transmission Control Protocol guarantees that if there was a network connection between two devices any message sent using it would arrive (Snader, n.d.).

4 Technical Development

4.1 System Design

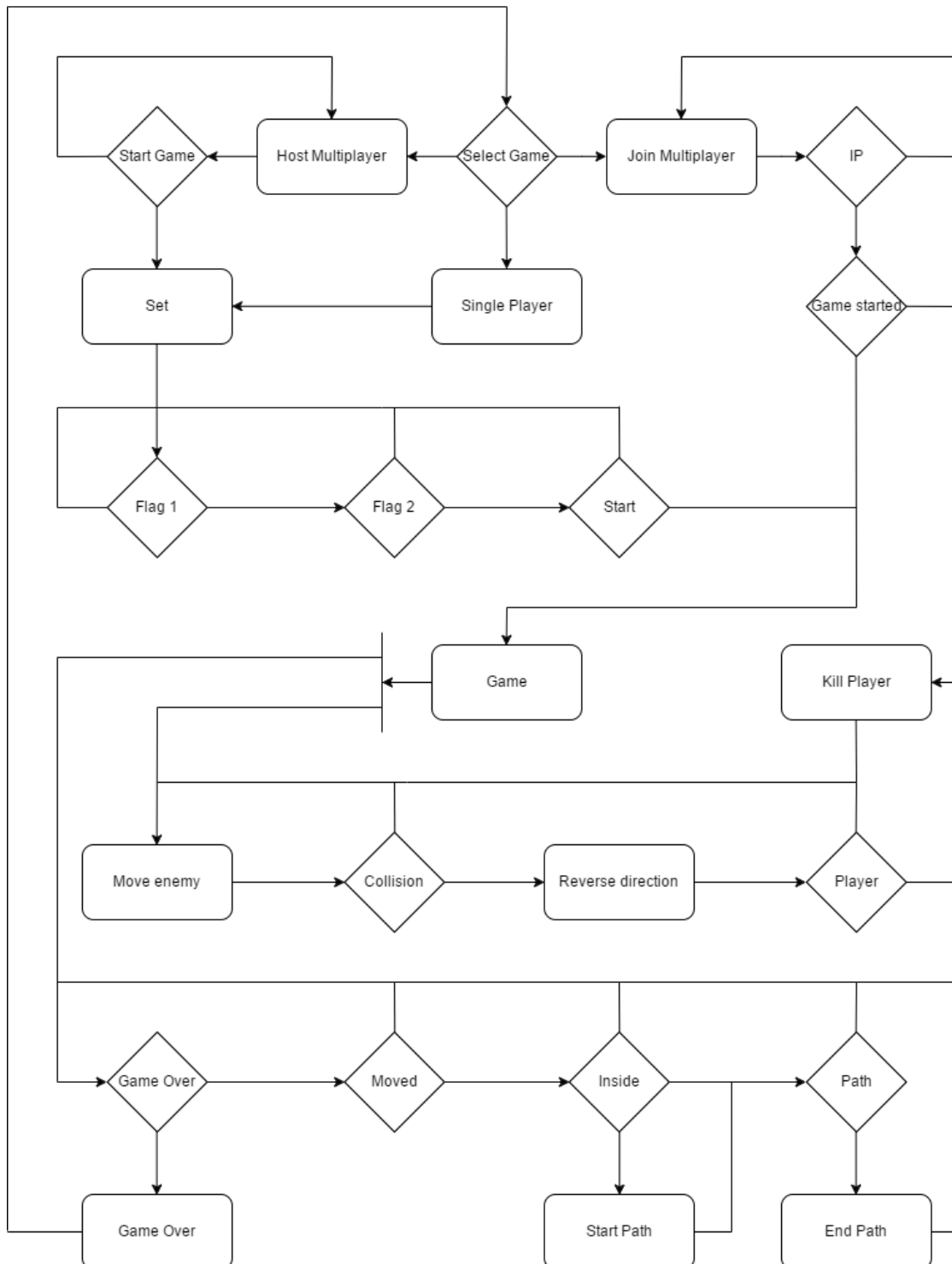


Figure 14: This image shows a graphical representation of the workflow and a stepwise diagram of activities and actions of the system.

The diagram above (Figure 14) is a UML activity diagram for the actions that could be taken while using the Capture the Campus! application.

Firstly, as it can be seen the diagram has no start or end, this is because the application itself once started could be used indefinitely unless terminated. The entry point into the diagram or program is at the top where the game over rectangle re-joins the game selection diamond.

In this diagram rectangles represent steps to be carried out while diamonds represent choices, the first choice represents the main menu with each branch and rectangle coming from the game selection diamond representing a button on the menu. The lines on the diagram represent the flow of logic, from any object each line can be traced from an end without an arrowhead to only one end with an arrowhead even if the lines converge. This shows the flow of logic from cause to effect.

If the single player menu button is selected then the game state changes to the set activity. However, if the host multiplayer button is selected then the game state first changes to the host multiplayer activity, in this activity the servers to host a game are started and connections from client devices, hoping to join a multiplayer game are accepted, this is represented on the diagram by the start game diamond. The only way to change the game state would be for the host player to select the start game button on their device and to change their own state to the set activity state.

Once in the set activity state two opposing corners of the play area also called flags are required to be selected to form the bounds of a square or rectangular initial play area, this is performed by first selecting one corner and then selecting the other before clicking the start game button. As can be seen on the diagram above, flag 2 cannot be selected before flag 1 and the game cannot be started before both flags are selected.

On the other end of the spectrum if the join multiplayer button is selected then the game state moves to wait to receive an IP address, this can occur in one of two ways, either the client on the join multiplayer device receives, verifies and then captures the host multiplayer device's IP address from a UDP broadcast or the IP address is manually entered into the join multiplayer device. For this reason, the IP address of the host multiplayer device is displayed on the host multiplayer activity screen and a button is located on the join multiplayer activity screen which once selected generates an alert dialog with a text entry field into which the host multiplayer device's IP address can be entered.

Once the join multiplayer device has received the host multiplayer device's IP address, it sits and waits for a signal from the host multiplayer device to show that the host multiplayer device has completed the set activity and the game can commence.

The game section has two threads of logic, this is represented by the vertical bar coming from the game rectangle that has two arrows exiting it, the top shorter path represents the logic for the enemy while the bottom longer path represents the logic for the player.

For the enemy, first the enemy's current position is saved before a new position is found using the wandering algorithm. The following diamond shows that the enemy is tested at its new position to see if it intersects or collides with anything, if it doesn't then the enemy is moved again. However, if the enemy has collided with an object then the variable used to contain the enemy's current velocity is changed to reflect this collision and the enemy is moved back to its previous position. The enemy is moved back because otherwise there is a possibility that the enemy could become stuck on the object that it has collided with.

If the enemy did collide then the diagram goes on to show that the object that was collided with is tested to see if it is a player, if it isn't a player then again, the enemy is moved and the

cycle continues. However, if it is a player then the player is effectively killed, meaning that the player's current path through the play area is removed and the player's score is reduced by half. This cycle continues until the game ends, which is represented in the other thread of logic.

On the other thread of logic, first the game state is checked to see if the requirements to end the game have been met. The end conditions for the game are that at least 75% of the initial play area has been captured when the area taken by all players is added together. If the game did not end then the position of the current player is checked to see if it is different from the last known location of that player, if it is not different then there is no point moving the representation of that player on the map or broadcasting their current position to other players and the logic moves on to check for the next player.

However, if the player did move then as shown in the diagram it goes on to check if the player is inside the play area, if the player is not inside the play area then the program continues to check if the player has a path associated with them, if they do it can be assumed that the player has just crossed from inside the play area to outside the play area. Thus, the program finds the point where the player crossed from within the play area to outside the play area by taking the line segment represented by the player's current position and previous position and finding the point where that line segment intersects the play area. After adding that point to the player's path the program, using the polygon clipping algorithm, finds the smallest of the two polygons that are created when the play area is bisected by the player's path and removes this polygon from the play area. The area of this polygon is then added to the player's score and removed from the variable tracking the current play area's area. This change is then broadcasted to all other player's

If the player is inside the play area the diagram shows that the program checks to see if a path has already been created, if a path has been started the player's current position is added to the list of the path's vertices. However, if a path has not yet been created then the program finds the point where the line segment represented by the player's current position and previous position intersects the play area, this position can be assumed to be where the player crossed into the play area and as such where the player's path through the play area should begin, this works the same as described for adding the last point to the player's path. If the program cannot find an intersection between these two line segments then it can be assumed that the player has just had their path removed by the enemy and as such no path is added to the player.

The program will continue to execute these two threads of logic for each player in the current game until a flag is raised by the end game conditions, when this occurs the program moves onto the score or game over activity which displays the player's score and offers to return them to the main menu to start the cycle again.

A full graphical overview of the entire system can be seen below (Appendix C:).

A full breakdown on how the user can interact with the system including images and diagrams showing the user interface can be seen below (4.2).

4.1.1 Networking Design

The diagram below (Appendix D:) shows a graphical representation of the solution for the networking component of the application, this diagram is called a code map.

This code map shows both the projects of the solution and classes contained within these projects which when compiled create the networking component of the application.

Included in the code map are arrows that depict inheritance between classes, objects being called, fields that are read from and written to and references from one class to another. In this instance, green arrows represent inheritance, purple arrows represent object calls, blue arrows represent field reads or writes and grey arrows represent references.

The full legend for the code map can be seen below (Appendix B:).

As can be seen in the code map below (Appendix D:) in the solution for the Networking component of the application there are nine projects, these will be discussed in the sections below.

Most of the classes contained within the projects of this solution and their contents are private.

This means that the contents of the solution's classes are only accessible from within the classes of the solution, this is because the contents of these classes have no need to be edited from anywhere but from within themselves.

Being private allows for the application of encapsulation meaning that the value or state of structured data within the class is hidden from the user when they are outside of the class. This is useful as it blocks unauthorised access to sensitive information held within the class, information such as a user's location (Noble, 2002).

Encapsulation also stops fellow developers, who may not fully comprehend an implementation, from breaking fundamental aspects of the underlying system (Berard, 2015).

4.1.1.1 Client Design

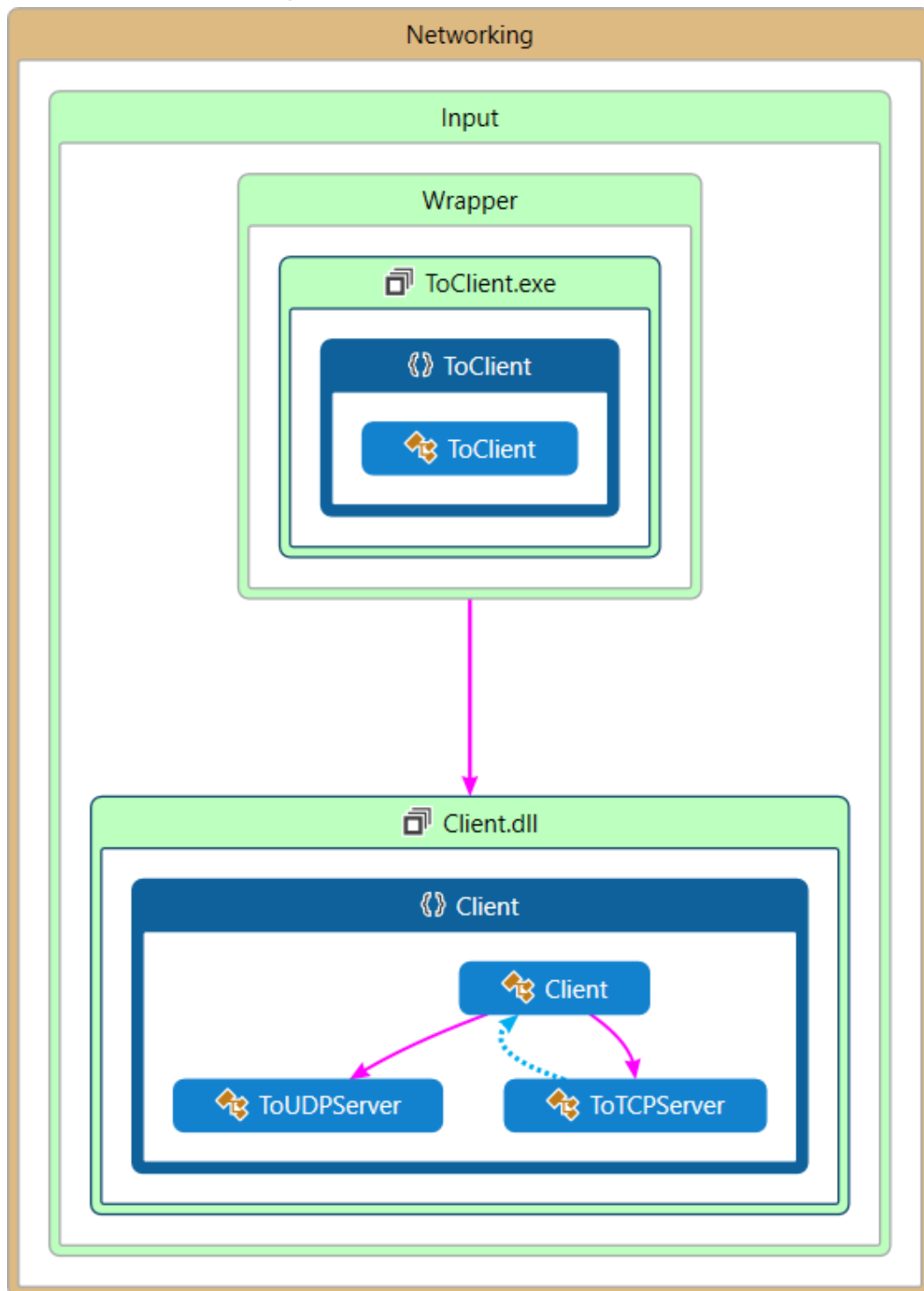


Figure 15: This image shows a diagram that describes the structure of the Client subsystem by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Wikipedia, 2016).

As can be seen in the code map above (Figure 15) in the solution for the Client there are two projects; an application called ToClient contained within a folder named Wrapper and a

library called Client, both projects are themselves contained within a folder called Input which itself is contained within a folder named Networking.
To see this solution in the context of other solutions within the Networking folder see below (Appendix D:).

The Client library has been designed in a modular fashion so that it can be adapted to work in any solution to send either UDP or TCP messages.

The Client library contains three classes:

The first class takes an input from the user, parses it and then passes the relevant information onto one of the other two classes in the project, this class is called the Client class.

Within the Client class is a public method named Input, the Input method is the method that should be called whenever a network communication is to be sent. To call this method an array of strings must be passed to it as an argument, the array of strings must at least contain a flag denoting the protocol to be used and the identifier of the information to be received or the identifier of the information to be sent and the piece of information.

The order of the strings in the string array does not necessarily matter; however, a piece of information must always follow its flag.

The flags that can be used to control the behaviour of the method include; “-t” which denotes that the message should be transmitted using the TCP protocol, “-u” which denotes that the message should be transmitted or received using the UDP protocol, “-i” which denotes that the following string in the string array is the IP address that the message should be sent to or received from and “-p” which denotes that the following string in the string array is the port that the message should be sent to or received from.

The second class takes the output from the Input class, sends it via the TCP protocol to a socket over the network defined by the IP address and the port specified by the input string and then waits a suitable amount of time for a relevant response from the recipient, this class is called the ToTCPServer class.

If the argument string from the Input class contains both an identifier and a piece of information the ToTCPServer class will immediately return with an acknowledgement that the message has been sent after sending its message. However, if the argument string from the Input class contains only an identifier the ToTCPServer class will wait for a response and then return that, if the ToTCPServer class does not receive a response after three seconds an appropriate error message is returned instead.

The third class receives a message broadcasted via the UDP protocol on the port specified by the input string on the local network. The third class waits a suitable amount of time for a relevant message, if the third class does not receive a response after three seconds an appropriate error message is returned. This class is called the ToUDPServer class.

The ToClient application is a test wrapper for the Client library. The ToClient application has been designed to allow arguments for the Client library to be built from the command prompt for rapid testing and prototyping.

The ToClient application contains one class:

This class takes an input from the user, constructs a string array from it and then passes this string array as an argument to the public Input method of the Client library, this class is called the ToClient class.

The command line argument to this application is a series of strings separated by spaces, the string must at least contain a flag denoting the protocol to be used and the identifier of the information to be received or the identifier of the information to be sent and the piece of information.

A useful advantage of the ToClient application test wrapper is that it allows for the autonomous testing of the Client libraries functionality through the piping of inputs from a text file to the post build arguments section of the application (Jeremy, 2014).

4.1.1.2 Watchdog Design

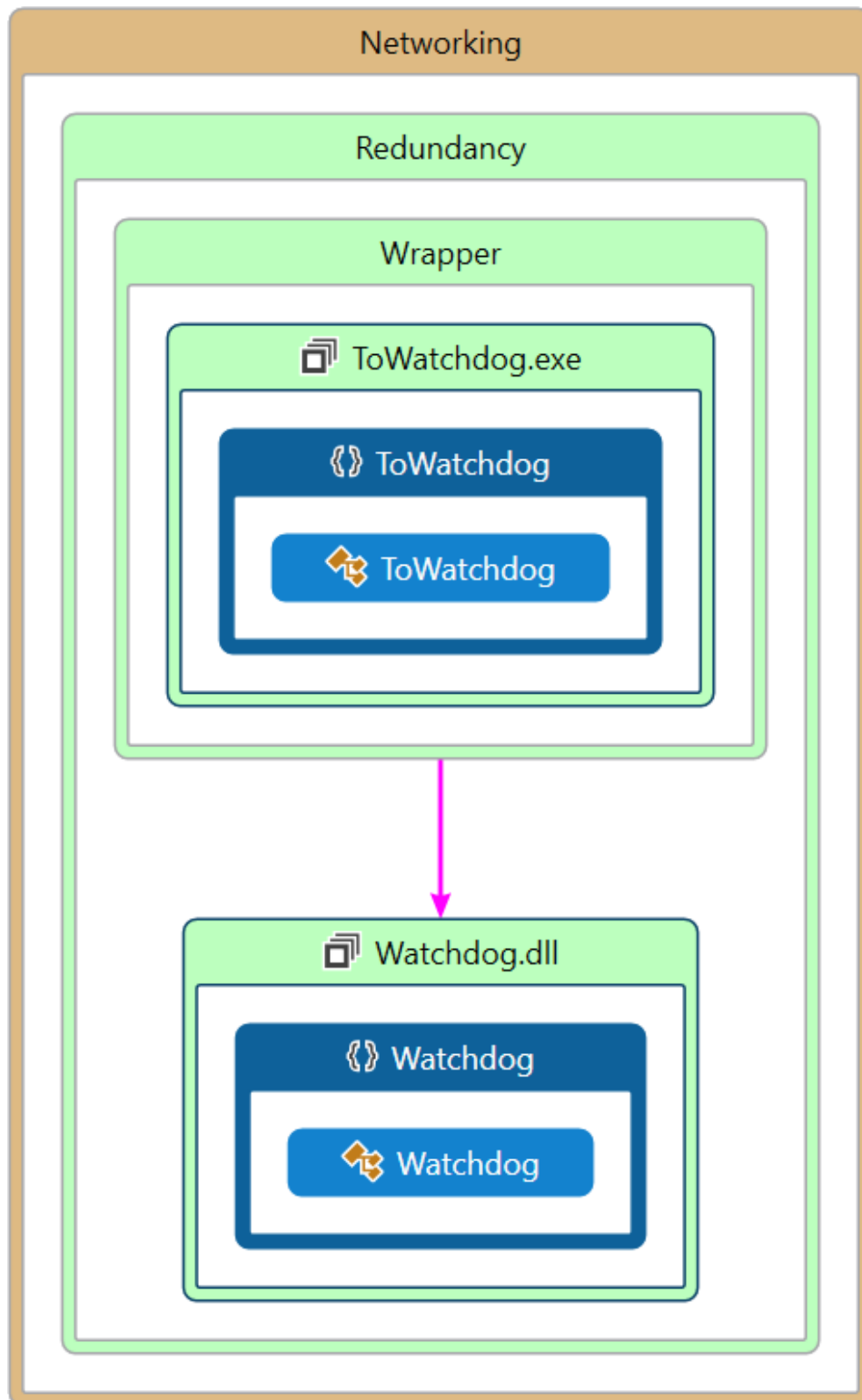


Figure 16: This image shows a diagram that describes the structure of the Watchdog subsystem by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Wikipedia, 2016).

As can be seen in the code map above (Figure 16) in the solution for the Watchdog there are two projects; an application called ToWatchdog contained within a folder named Wrapper and a library called Watchdog, both projects are themselves contained within a folder called Redundancy which itself is contained within a folder named Networking. To see this solution in the context of other solutions within the Networking folder see below (Appendix D:).

The Watchdog library has been designed in a modular fashion so that it can be adapted to work in any solution as a watchdog or heartbeat to a server.

The Watchdog library contains one class:

This class starts a task which starts an instance of the TCPServer project. The class then polls the instance of the TCPServer repeatedly for the time of the last communication from the application that is using the instance of the TCPServer, this application should be made to update the server with the current time, every second, for this application to poll. If the time of the last communication from the application using the instance of the TCPServer is beyond a certain threshold the class cancels the task on which the instance of the TCPServer is running using a cancellation token.

The class cancels the task because if the application hasn't updated the timer within the threshold it can be assumed that the application has crashed or ended and thus the task on which the TCPServer is running needs to be cancelled to prevent it running indefinitely.

This class is called the Watchdog class.

The Watchdog class could be likened to a mixture of a watchdog application and a heartbeat application.

A watchdog application is an electronic timer which during normal operations is reset by itself, if the timer elapses corrective actions are taken to resolve the issue which caused the timer to elapse in the first place. Watchdog applications are usually used in situations where humans cannot access the device on which the application is running or if it would be impossible to react to the issues in a reasonable time (IBM, n.d.).

A heartbeat application is a periodic signal generated by the application which is used to show normal operation (R, 2013).

The Watchdog class uses aspects of both applications, it takes influence from watchdog applications as it uses a timer which when elapsed corrective actions are taken and it takes influence from heartbeat applications as the timer is updated by a periodic signal generated by the application.

The ToWatchdog application is a test wrapper for the Watchdog library. The ToWatchdog application has been designed to allow the Watchdog library to be started from the command prompt for rapid testing and prototyping.

4.1.1.3 TCPServer Design

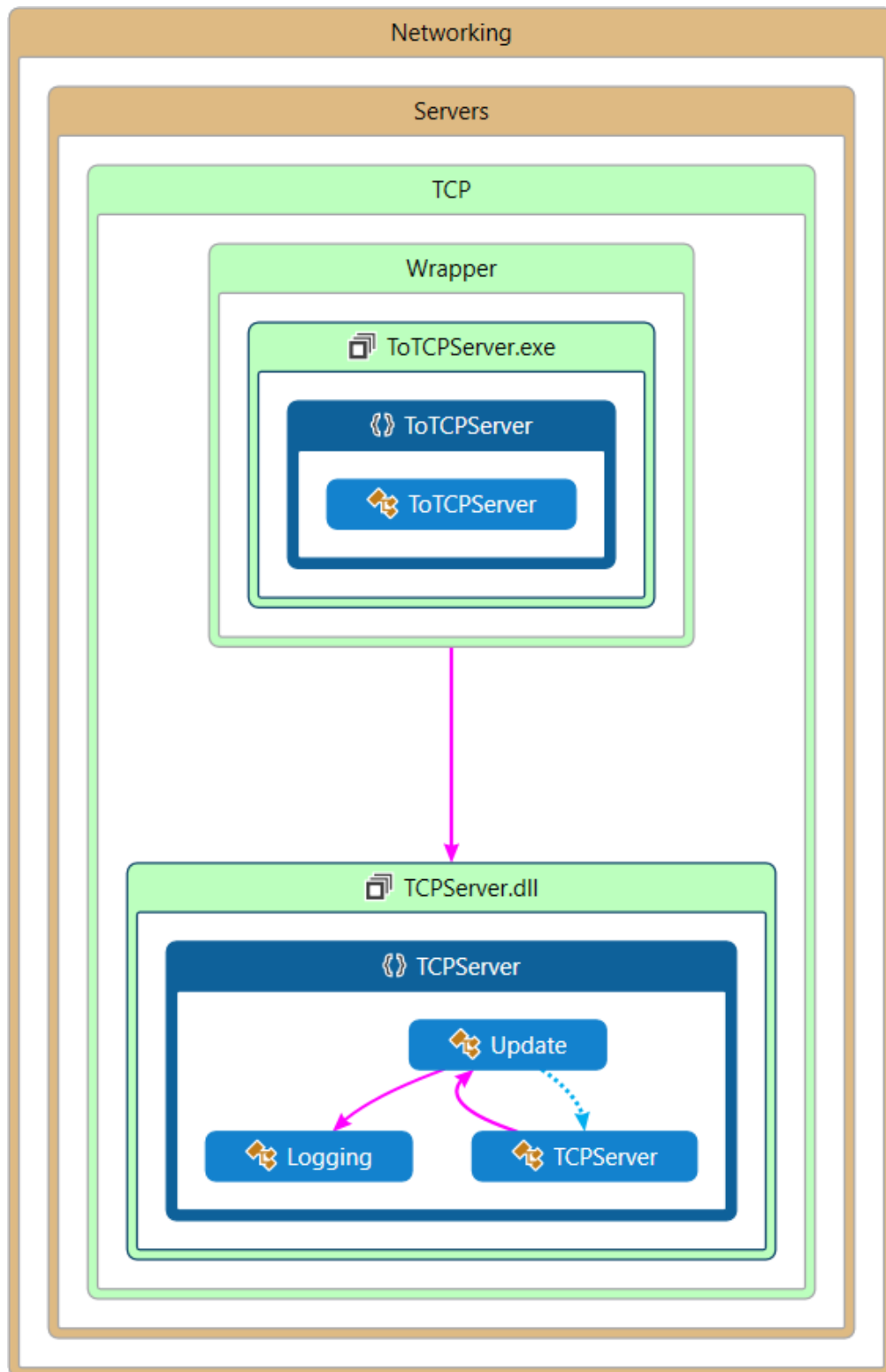


Figure 17: This image shows a diagram that describes the structure of the TCPServer subsystem by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Wikipedia, 2016).

As can be seen in the code map above (Figure 17) in the solution for the TCP server there are two projects; an application called ToTCPServer contained within a folder named Wrapper and a library called TCPServer, both projects are themselves contained within a folder called TCP which itself is contained within a folder named Servers all of which are contained within a folder called Networking.

The TCPServer library has been designed in a modular fashion so that it can be adapted to work in any solution to receive TCP requests.

The TCPServer library contains three classes:

The first class starts the TCP server, this class contains a loop which on each iteration checks for a socket connection, if a request is being sent to this socket it then creates a thread to deal with the request, each thread is added to a thread pool to keep track of all current threads in case an error occurs. This class is called the TCPServer class.

The second class is the class which each thread from the TCPServer class is started on. If the request contains both an identifier and information then the thread will save the information to a concurrent dictionary and return an acknowledgement that the information was stored correctly. However, if the request contains only an identifier then the thread looks up the identifier in the concurrent dictionary, if there is an entry in the dictionary then the thread returns the information stored under the identifier, if the concurrent dictionary does not contain the identifier then the thread returns an error. This class is called the Update class.

The ToTCPServer application is a test wrapper for the TCPServer library. The ToTCPServer application has been designed to allow the TCPServer library to be started from the command prompt for rapid testing and prototyping.

4.1.1.4 UDPServer Design

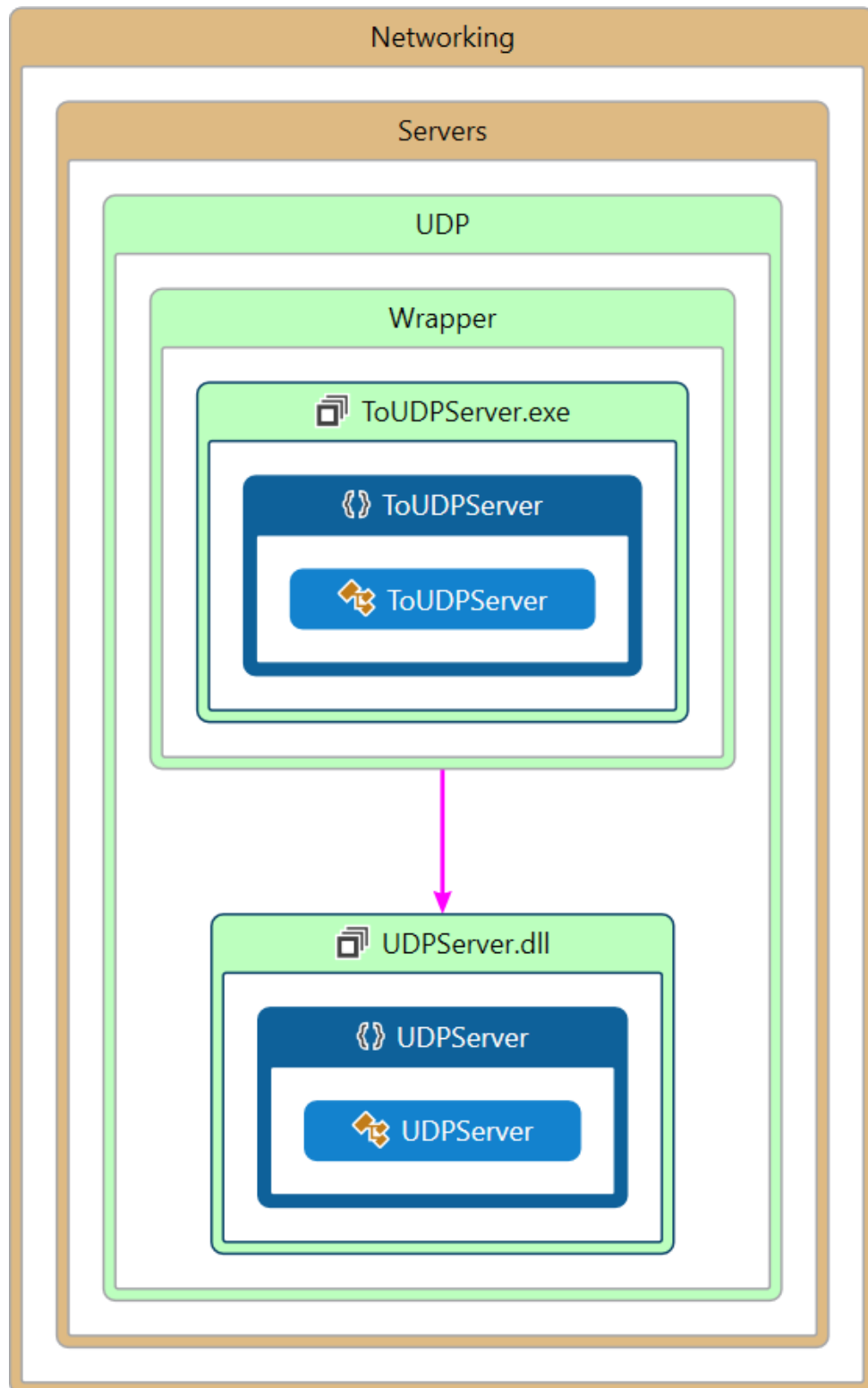


Figure 18: This image shows a diagram that describes the structure of the UDPServer subsystem by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Wikipedia, 2016).

As can be seen in the code map above (Figure 18) in the solution for the UDP server there are two projects; an application called ToUDPServer contained within a folder named Wrapper and a library called UDPServer, both projects are themselves contained within a folder called UDP which itself is contained within a folder named Servers all of which are contained within a folder called Networking.

The UDPServer library has been designed in a modular fashion so that it can be adapted to work in any solution to send or receive UDP requests.

The UDPServer library contains one class:

This class simply sends a UDP broadcast that contains the servers IP address and a coded message, the coded message can be used to verify the UDP broadcast, if the broadcast can be verified as coming from this server then the IP address contained within the broadcast can be used to communicate with any other servers running from the same device. This class is called the UDPServer class.

The ToUDPServer application is a test wrapper for the UDPServer library. The ToUDPServer application has been designed to allow the UDPServer library to be started from the command prompt for rapid testing and prototyping.

4.1.2 CaptureTheCampus Design

The diagram for the CaptureTheCampus application as seen below (Appendix E:) shows a graphical representation of the solution for the CaptureTheCampus application, this diagram is called a code map.

This code map shows both the namespaces of the solution and classes contained within these projects which when compiled create the Capture the Campus! application. The full legend for the code map can be seen below (Appendix B:).

As can be seen in the code map below (Appendix E:) in the solution for the CaptureTheCampus application there are six namespaces, these will be discussed in the sections below.

Most of the classes contained within the namespaces of this solution and their contents are private.

Being private allows for the application of encapsulation (Noble, 2002).

4.1.2.1 Game Design

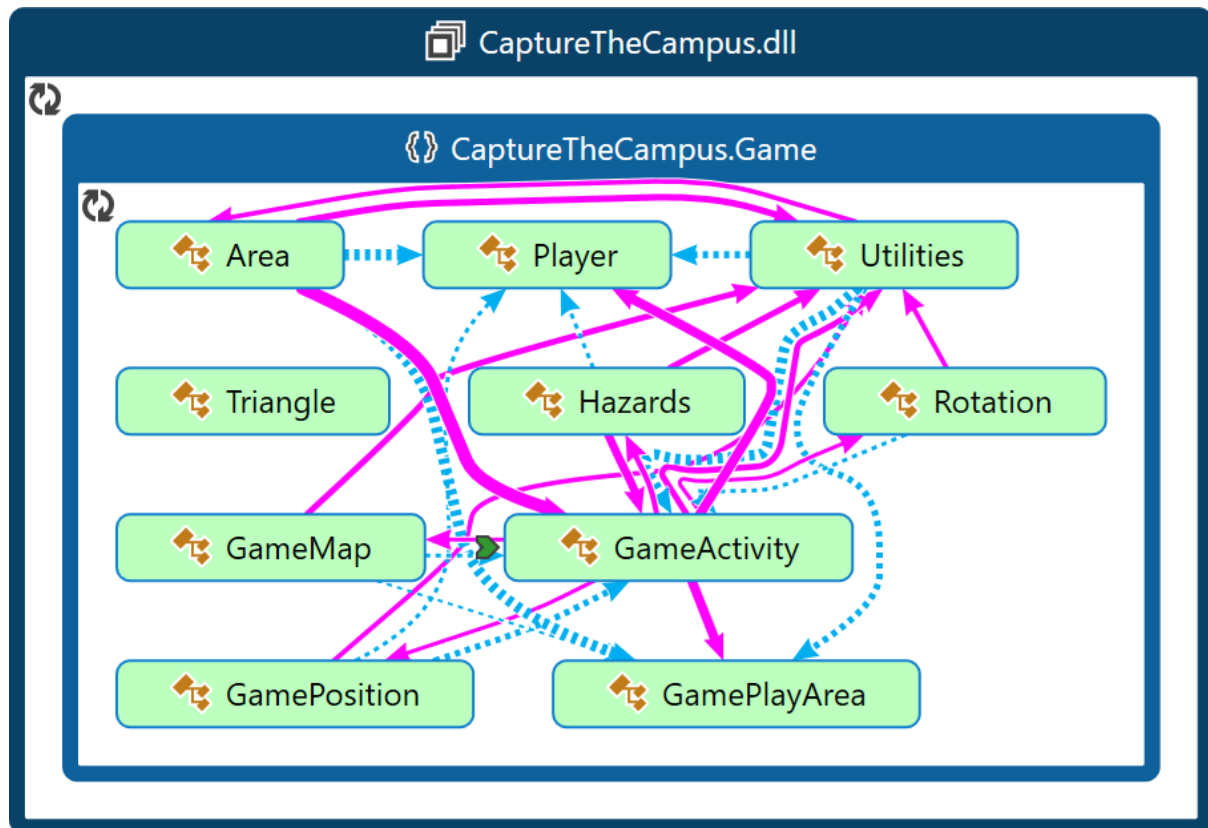


Figure 19: This image shows a diagram that describes the structure of the Game subsystem by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Wikipedia, 2016).

The diagram for the Game namespace as seen above (Figure 19) shows a graphical representation of the namespace for the gameplay functionality of the CaptureTheCampus application.

The Game namespace contains ten classes:

The GameActivity class is called to start the game, the specific behaviour of the GameActivity is dictated by the intent that it is called from. The intent contains both the vertices of the initial play area and the string that dictates what mode of game is to be played, if the string contains single player then a single player game is to be played, if it contains host then the game is a host of a multiplayer game and if the string contains join then the game is the client of another multiplayer game.

Shortly after the game begins it initialises instances of both the Player class and the GamePlayArea class.

The Player class is an object that represents an instance of a player in the game, it contains; their score, their current location, a list to represent their path through the play area and several Boolean variables to represent their current state. For each player in a game another instance of this class is initialised and added to a list

The GamePlayArea class is an object that represents the current play area in the game, it contains; a list of vertices that represents the current play area, a list of polygons that are currently being drawn on the map and a Boolean flag which dictates if the play area has been drawn successfully. The list of vertices is populated by the vertices passed to this activity from the set activity via the intent.

The GameMap class is an interface for the OnMapReadyCallback view, this class is called after the GameActivity has been initialised. This class is used to start an instance of a map object inside a fragment on the GameActivity screen.

This class also adds the markers that represent the players to the map, the polygon that represents the play area to the map and the circle that represents the enemy to the map.

The GamePosition class is an interface for the ConnectionCallbacks, the OnConnectionFailedListener and LocationListener views, this class is called after the GameActivity has been initialised. This class is used to setup location requests at a suitable interval using the fused location API.

Roughly once every second the OnLocationChanged method of this interface is called which gets the location of the device from the GPS sensors and from the mobile network and checks to see if this position is different to the previous position of the device, if it is as stated above the program continues with updating the player's position using the Utilities class.

The Utilities class contains numerous functions which are used throughout the game activity. This includes functions which; construct a map, build and place markers on the map, update the location of players, move the camera, build and place polylines on the map, build and place polygons on the map, build and place circles on the map and find the points where a circle intersects a polygon.

The Area class contains numerous functions which are used to update paths and polygons. Specifically these functions include; functions which find the state of a player using their flags and then update their path accordingly, functions which find the initial and final point of a path where it intersects with a polygon, functions which amend paths which intersect, functions which reset the paths of players, functions which build new polygons from the points where a path bisects a larger polygon, functions which test these new polygons for their areas and remove these areas from the play area and functions which update the score of players.

The Hazards class contains the logic related to the operation of the autonomous enemy, this code is ran on a separate thread to the rest of the game logic and is updated roughly once a

40

second. The enemy first checks to see if it is inside the play area, if it is not it uses the polygon tessellation algorithm to keep finding new random locations within the play area until it is inside the play area. Once the enemy has found a location it sets itself a random trajectory which is represented as a number of degrees between zero and three hundred and sixty, then the enemy updates its location using the wandering algorithm before checking to see if it has intersected either the play area or another player.

If the enemy intersects with the play area wall the enemy resets itself to its previous position, adjusts its velocity to represent the fact it collided with the wall and the attempts to move itself again, if the enemy cannot move after 10 separate attempts it can be assumed that the enemy has become stuck and the enemy finds itself a new random location and trajectory. If the enemy intersects with a player's path it removes the player's path, which prevents the player from creating another path until they exit and re-enter the play area and sets a flag contained within that player's Player object to represent the fact that they have been killed.

4.1.2.2 Menu Design

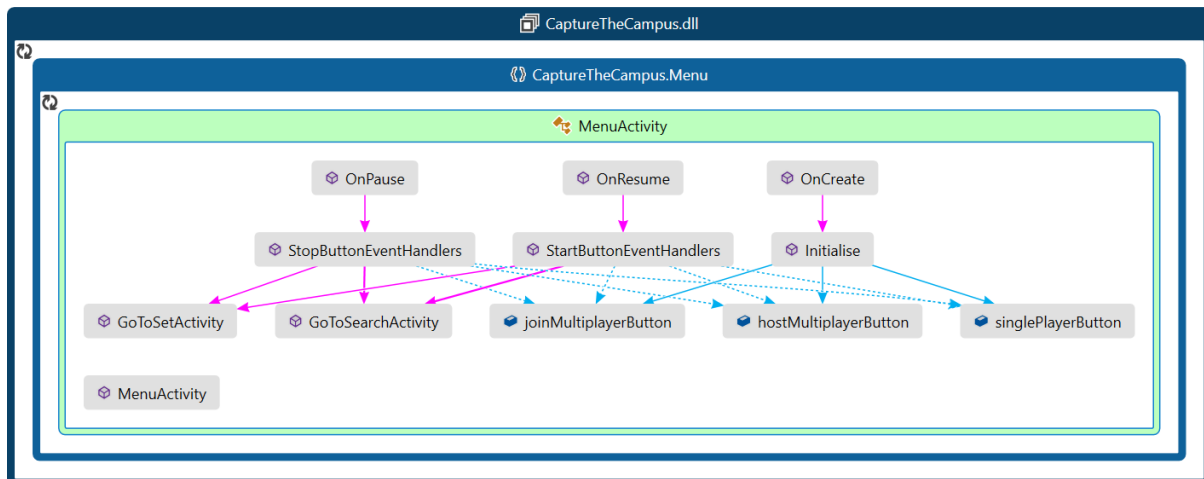


Figure 20: This image shows a diagram that describes the structure of the Menu subsystem by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Wikipedia, 2016).

The diagram for the Menu namespace as seen above (Figure 20) shows a graphical representation of the namespace for the menu of the CaptureTheCampus application.

The Menu namespace contains one class:

The MenuActivity class is called once the MainActivity class has verified that the device has the required libraries installed.

The MenuActivity contains three buttons which when pressed start different game modes, the top button starts the single player game by calling an intent to the set activity which contains a string that identifies it as a single player game instance. The next two buttons start the host multiplayer activity and join multiplayer activity respectively by calling an intent to the search activity which contains a string that identifies the activity as the correct instance.

4.1.2.3 Score Design

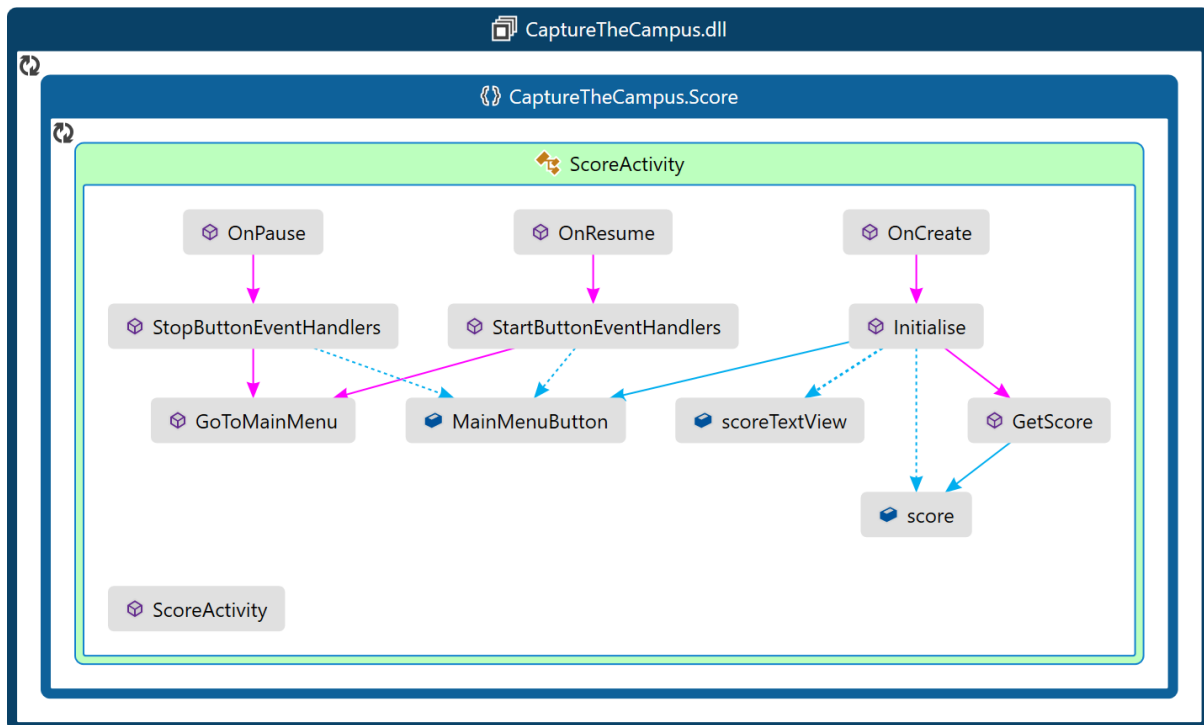


Figure 21: This image shows a diagram that describes the structure of the Score subsystem by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Wikipedia, 2016).

The diagram for the Score namespace as seen above (Figure 21) shows a graphical representation of the namespace for the score functionality of the CaptureTheCampus application.

The Score namespace contains one class:

The ScoreActivity class is called immediately when the end conditions of a game are reached. The score from the game is passed into the ScoreActivity using an Intent and is displayed on the screen above a large button which when pressed returns the player to the menu activity

4.1.2.4 Search Design

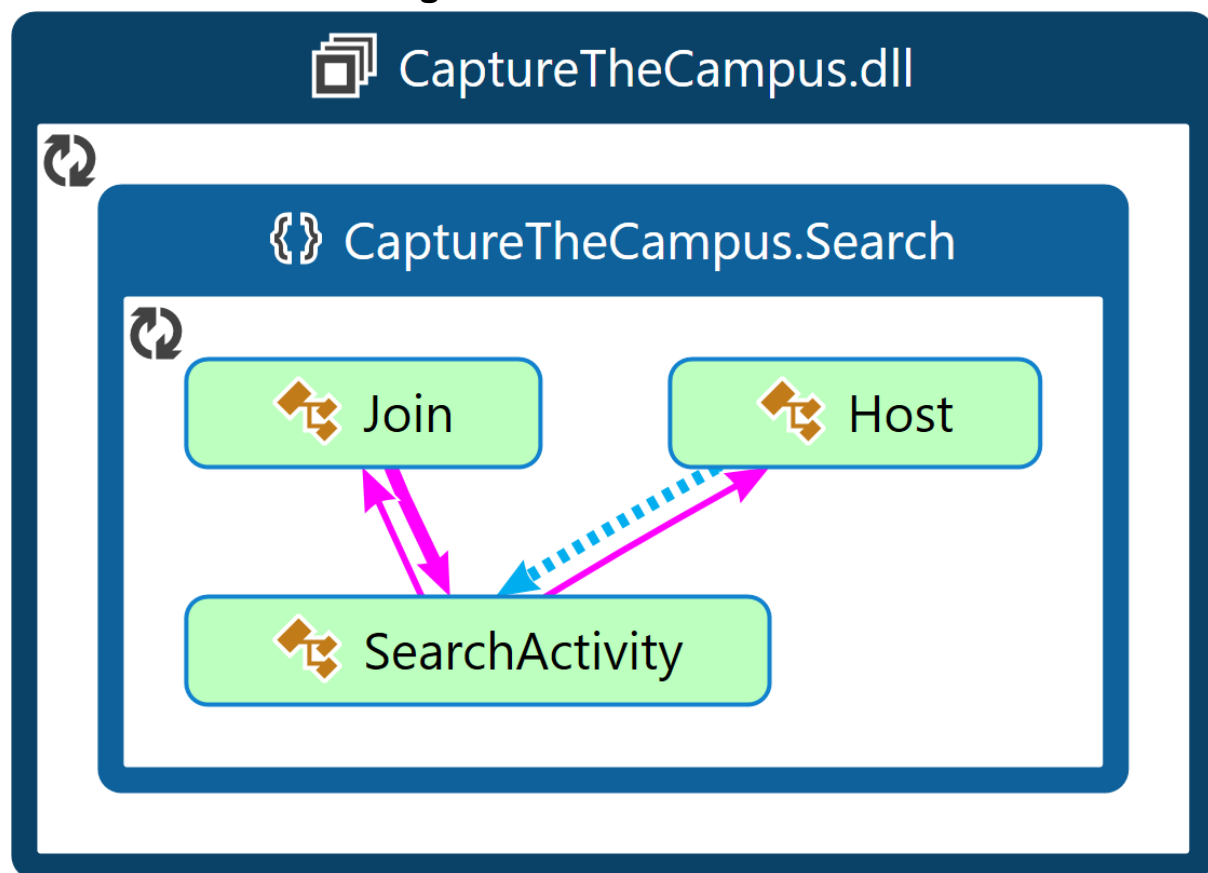


Figure 22: This image shows a diagram that describes the structure of the Search subsystem by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Wikipedia, 2016).

The diagram for the Search namespace as seen above (Figure 22) shows a graphical representation of the namespace for the search functionality of the CaptureTheCampus application.

The Search namespace contains three classes:

The SearchActivity is called whenever either of the multiplayer buttons are pressed in the menu activity. The exact behaviour of the SearchActivity is dictated by a string that is received from the Intent that is used to call the activity, if the string contains the word join then the activity behaves like a client and if the string contains the word host it behaves like a host.

The activity itself contains three objects, a textview at the top of the page which contains either the IP address of the device or a message dictating whether the device is connected to the server depending on if the device is a host or a client respectively, a large loading icon to reassure the user that the application has not crashed and a button which either starts the game or allows the user to input an IP address manually depending on if the device is a host or a client respectively.

If the device is a host then the Host class is also called, within this class the servers to establish an initial connection between host and client are started. This includes both an

instance of the UDP server and the Watchdog which itself starts and instance of the TCP server. The host broadcasts its own IP address via the UDP server with a secret key. The host initially enters a variable into the TCP server which contains the current number of players, it then goes on to continually poll its own TCP server to see if a client has incremented this counter, if a client has connected and adjusted this counter then the host assumes that another player has joined the game and adjusts itself accordingly. When the button is pressed to start the game the Host class passes a flag to the game activity that informs it that it is a host and passes the relevant information about the clients connected.

If the device is a client then the Join class is called instead, within this class a thread is started which initially continually searches for a UDP broadcast that contains the secret key, when one is found the IP address that the key was sent from is parsed and a TCP message is sent to the same IP address, if the TCP message is successfully sent then the Join class assumes that it is connected to a host. This method works very similarly if the IP address is entered manually.

Once the client is connected to a host it attempts to adjust a predefined variable contained within the hosts servers which dictates the number of players connected to the host, if the client can successfully change this variable then the client moves on to wait for a signal from the host to indicate that the multiplayer game has commenced.

4.1.2.5 Set Design

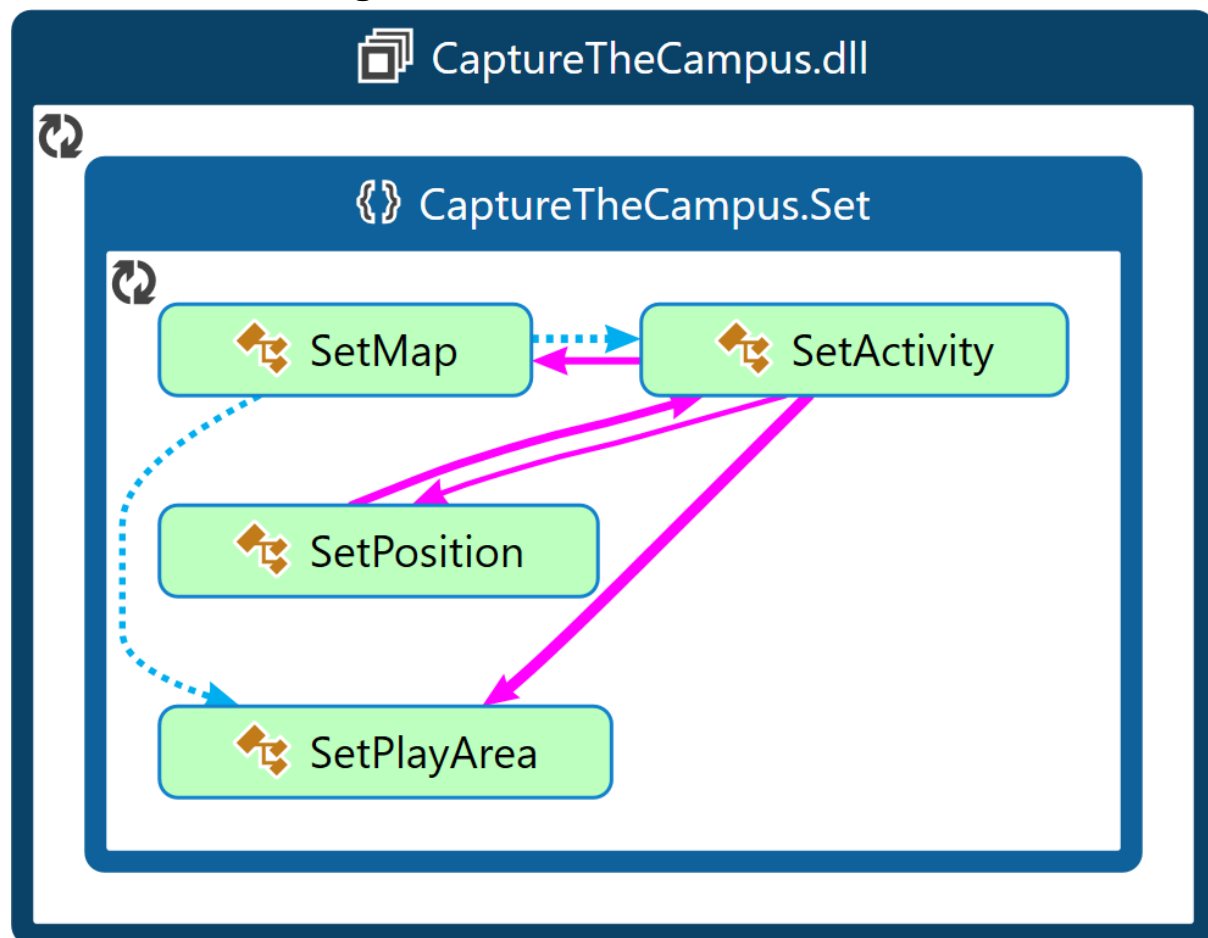


Figure 23: This image shows a diagram that describes the structure of the `Set` subsystem by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Wikipedia, 2016).

The diagram for the `Set` namespace as seen above (Figure 23) shows a graphical representation of the namespace for the set functionality of the `CaptureTheCampus` application.

The `Set` namespace contains four classes:

The `SetActivity` is basically a stripped-down version of the `GameActivity` which contains just enough functionality to operate to place markers on the map to represent the corners of an initial play area.

Like the `GamePlayArea` the `SetPlayArea` contains the information related to the play area polygon. However, unlike the `GamePlayArea` the `SetPlayArea` only contains an array of vertices and one polygon, this is because when the markers are placed upon the map to create the initial play area by selecting two opposing corners there will only ever be one polygon with four vertices, if a marker is moved the polygon is redrawn.

The `SetPosition` class contains the same logic as the `GamePosition` class but with fewer checks on the location that is returned, this is because the `SetPosition` class is only called once to get the initial camera location before it is disconnected.

4.1.2.6 Static Design

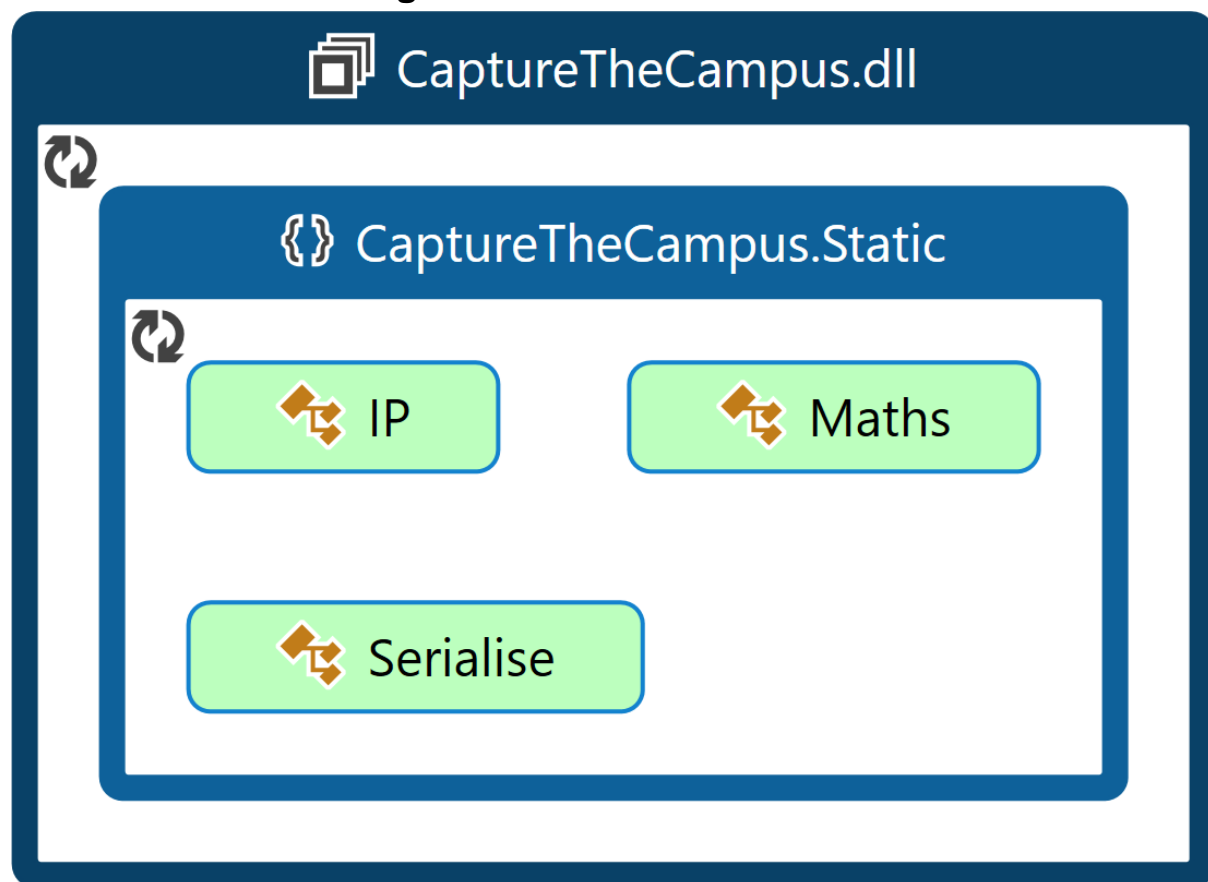


Figure 24: This image shows a diagram that describes the structure of the Static subsystem by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Wikipedia, 2016).

The diagram for the Static namespace as seen above (Figure 24) shows a graphical representation of the namespace for the utilities of the CaptureTheCampus application.

The Static namespace contains three classes:

The Maths class contains numerous mathematical functions which are used throughout the program. This includes functions which; find if a point is within the bounds of a polygon, find the point of intersection between two line segments if they intersect at all, find the area of a polygon, find the shortest line segment from a list of vertices, find the centre of a polygon, find the points of a line segment which is extended from the points of another line segment, find a random point inside a polygon, find if a circle and line segment have intersected and convert to and from degrees to a unit vector.

The IP class is used to find one of the IP address of the current device, this works by requesting the host address from the domain name server.

The Serialise class contains methods which are used when sending information to and from the server. One of the methods contained within this class is a method which converts a list of latitude and longitude objects of any length to a string of comma separated variables by

first converting all list objects to strings and then inserting a comma between the latitude and longitude of each list object before then inserting a comma between each list object. Another method is one which converts a string of comma separated variables into a list of latitude and longitude objects of any length by performing the same task as the previous method but in reverse.

4.2 User Interface Design

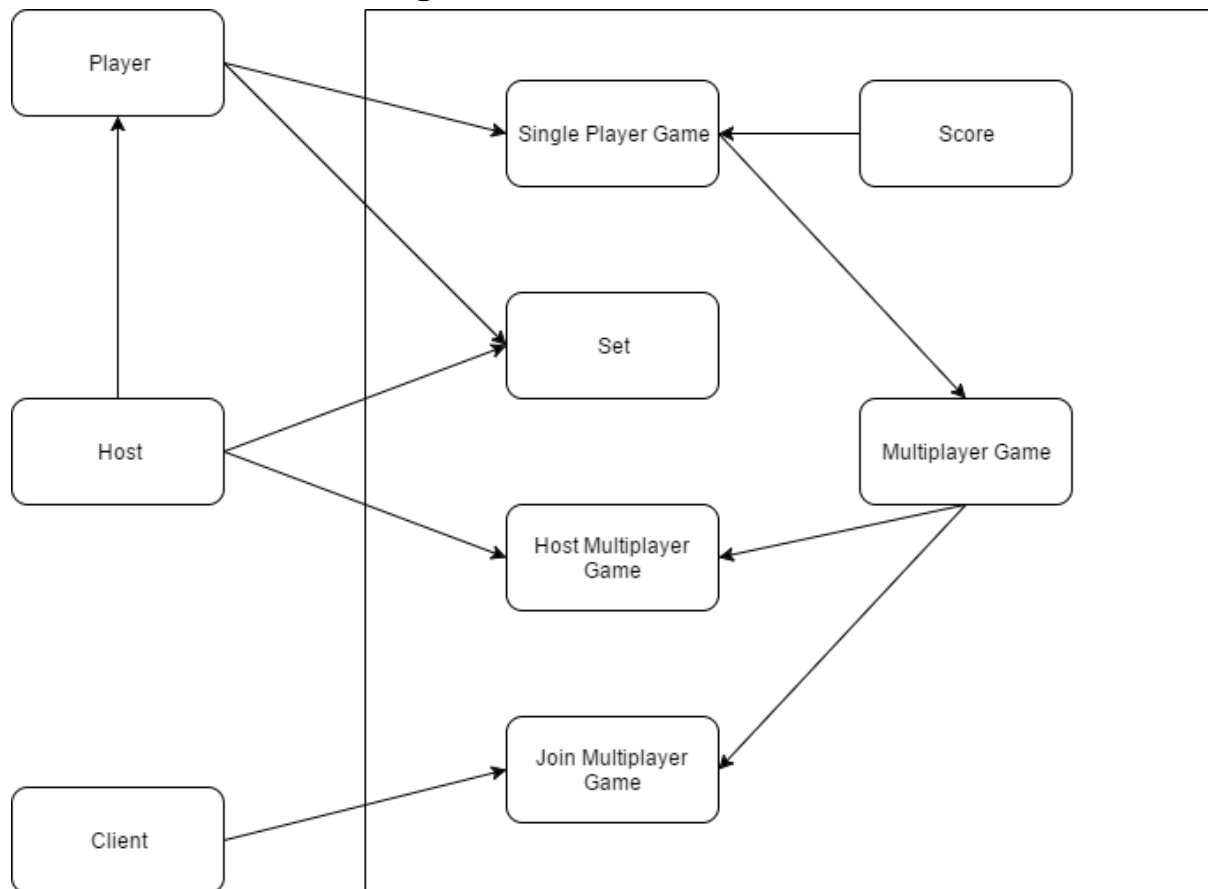


Figure 25: This image shows a representation of the simplest interactions a user can have with the system.

The diagram above (Figure 25) is a UML use case diagram for the interactions that different kinds of users or actors can have with the Capture the Campus! application.

The rectangles on the left outside of the larger box like rectangle on the right represent the different kinds of users or actors that could interact with the system. Any person could be any user at any given time and most people will be multiple different users depending on which game mode they are playing.

The top most user is the standard single player user, this user only has need to play a single player game and as such required access to the single player game, this required an option on the main menu to select this game mode.

Each instance of the game required an instance of the Score activity to display once the game has ended, this is represented by the inheritance arrow from the score rectangle to the single player game rectangle. The Score activity required its own option to return to the menu after the user has finished with that game instance.

For the single player game to operate a separate instance of the Set activity was required, this can be seen by the separate arrow coming from the player rectangle to the set rectangle. Obviously again the Set activity required its own user interface with a multitude of options to select flags to define the play area.

The second user is the host in a multiplayer game situation, this user required most of the same functionality and user interface options as the single player user, hence the inheritance arrow to this user, but also required another activity called the Host activity to start the servers associated with a multiplayer game and to connect to and synchronise the client users.

This activity required its own user interface which would include the option to start the multiplayer game and display the IP address of the device hosting the multiplayer game instance.

The final user is the client in a multiplayer game situation, this user required comparatively less functionality to the single player user and the host user, this user required an activity called the Join activity to connect to and synchronise their device with the host user. Once synchronised this user did not require the same functionality as the host user because a lot of the processes which would normally be performed by a single player user in a single player game could be performed by the host multiplayer user for the clients. Thus, the client user did not require any further user interfaces. However, for the sake of redundancy in case the UDP client or server fails to acquire the IP address of the host device the option to manually enter the IP address of the host device was required.

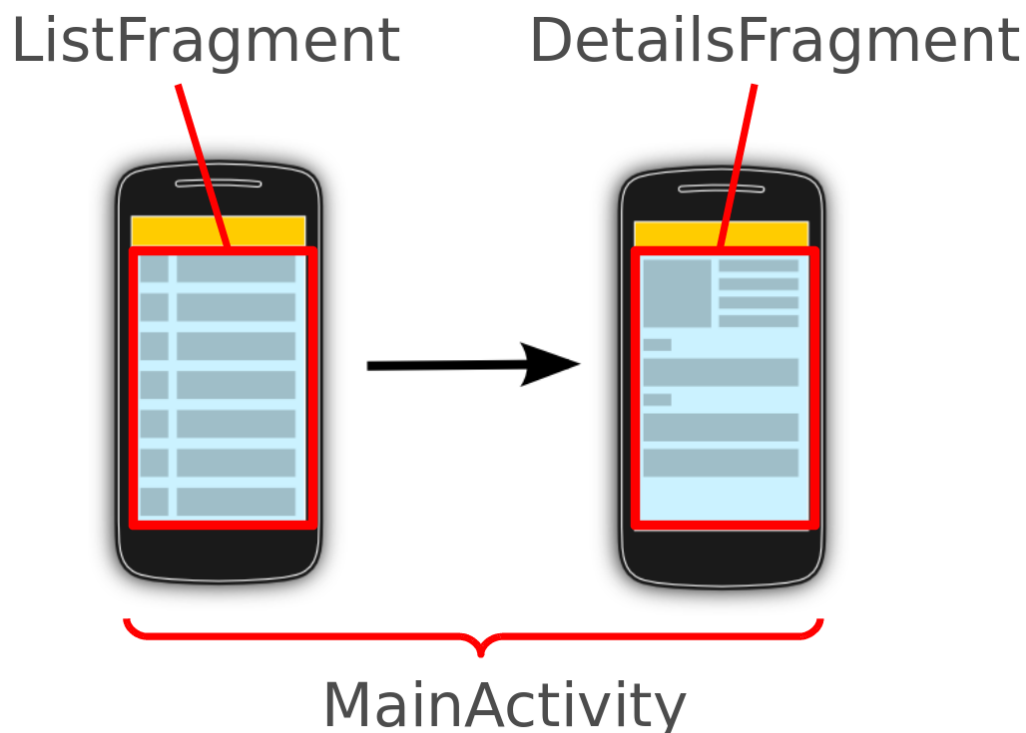


Figure 26: This image shows two examples of menu styles used in mobile application (Vogel, 2016)

The image above (Figure 26) shows two representations of menu styles generally used in mobile applications, design ideas were taken from this image to aid in the visualisation of the projects user interface.

For this project a main menu was required to select the game mode to be played. Either; single player, host multiplayer or join multiplayer.

There are two main mobile application user interface design ideologies, one which states that menus should follow list like structures called ListFragment and one that states that

menus should follow clusters of information called DetailsFragment (Balagtas-Fernandez, et al., 2009) (Vogel, 2016).



Figure 27: This image shows examples of the user interface for the Android launcher Arrow Launcher by Microsoft (Funk, 2015)

The image above (Figure 27) shows an example of both types of menu user interface being used for different reasons in the same application. On the left, there is an example of a DetailsFragment menu with boxes representing objects within the menu and in the centre and on the left there are ListFragment menus showing ordered lists.

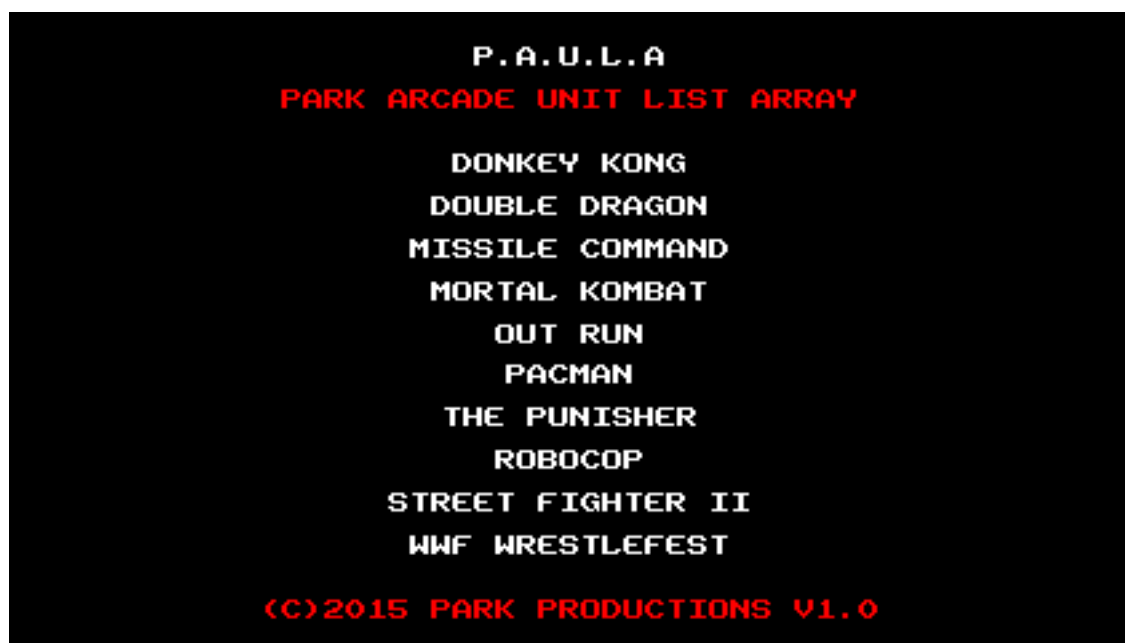


Figure 28: This image shows an example of a retro arcade style menu (Park Productions, 2015)

To further the emulation of Qix and other retro style arcade games in this project a ListFragment style ordered list menu was adopted, an example of this can be seen in the image above (Figure 28).

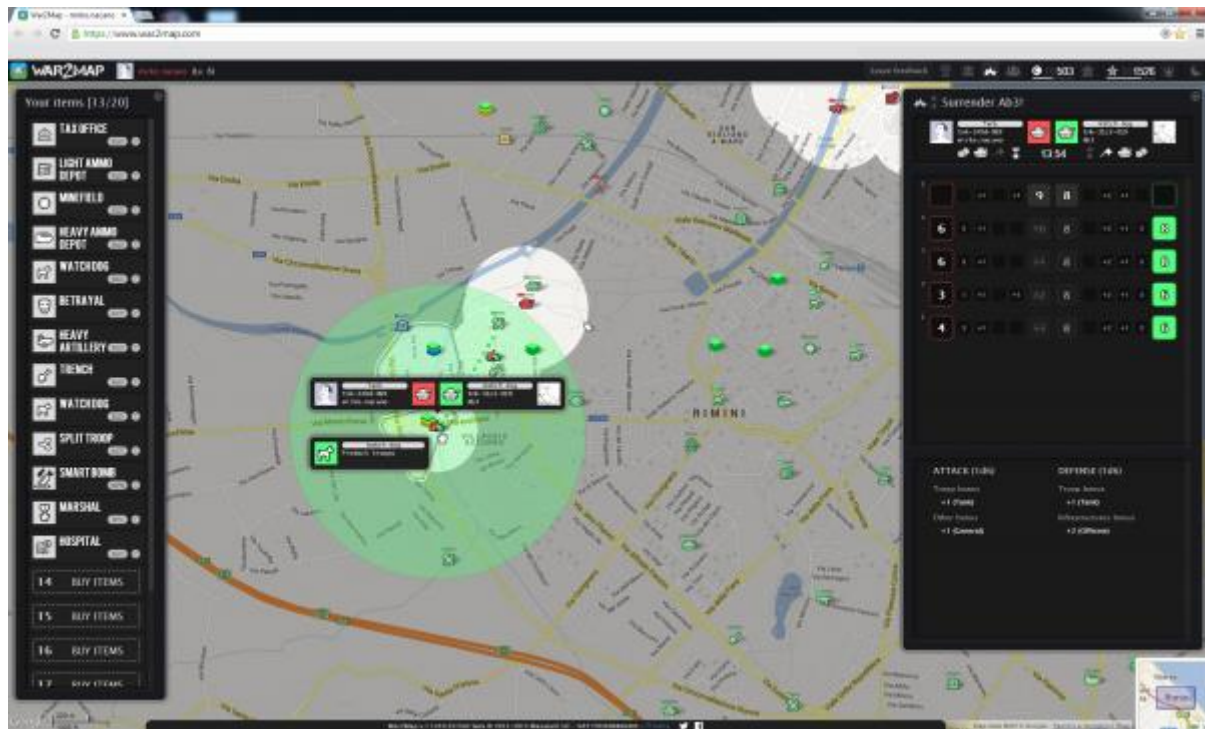


Figure 29: This image shows a standard game screen for the Google Maps API based game War2Map (Dempsey, 2013)

The image above (Figure 29) shows a representation of a game being played using the Google Maps API, design ideas were taken from this image to aid in the visualisation of the projects user interface.

Firstly, dotting the map there are multiple instances of tokens representing game objects, this showed that the Google Maps API offered the function to draw markers upon the map. The ability to draw markers over the map was used in this project to represent player characters and non-player entities.

Secondly, a large green shape can be seen, this showed that the Google Maps API offered the function to draw polygons upon the map. The ability to draw polygons over the map was used in this project to represent the play area and to represent the areas of the play area controlled by a player.

Thirdly, a dotted polyline can be seen spanning the distance between two avatars, this showed that the Google Maps API offered the function to draw polylines connecting two coordinates upon the map. The ability to draw polylines connecting two coordinates over the map was used in this project to represent the paths taken by each player character crossing the playing area.

Finally, it was assumed that the map retained its panning and scaling features.

The images below show the final user interface of the Capture the Campus! application (Figure 30) (Figure 31) (Figure 32) (Figure 33) (Figure 34) (Figure 35) (Figure 36) (Figure 37) (Figure 38).

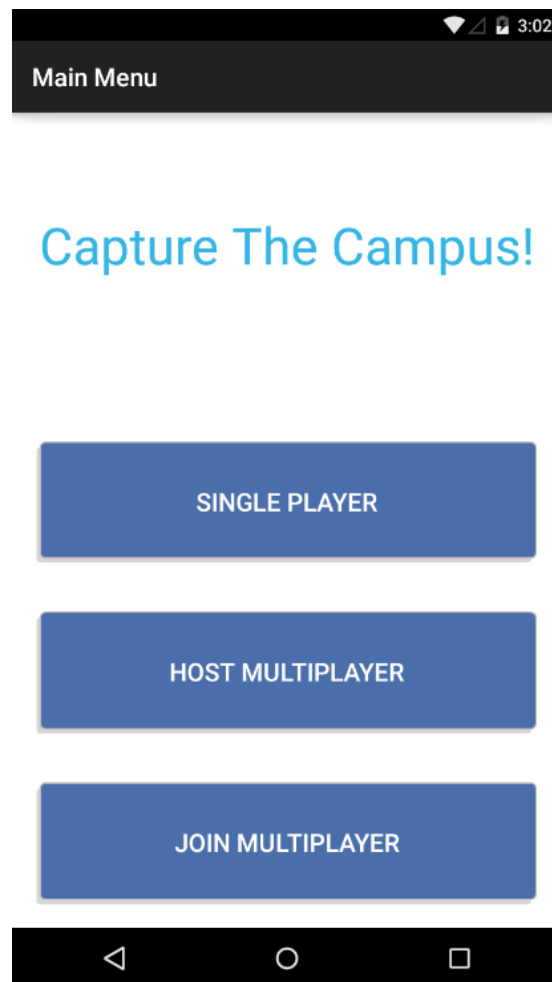


Figure 30: This image shows the user interface for the Menu activity.

The image above (Figure 30) shows the final user interface for the Menu activity. As can be seen the final user interface in this instance made use of the ListFragment style ordered list methodology, as discussed previously, to create the options for each game mode.



Figure 31: This image shows the user interface for the Set activity before any markers or flags have been set.

The image above (Figure 31) shows the final user interface for the Set activity before any markers or flags have been placed upon the map to define the initial play area. As can be seen the final user interface in this instance made more of a use of the DetailsFragment methodology as it made more sense in this instance for the menu to be less intrusive.

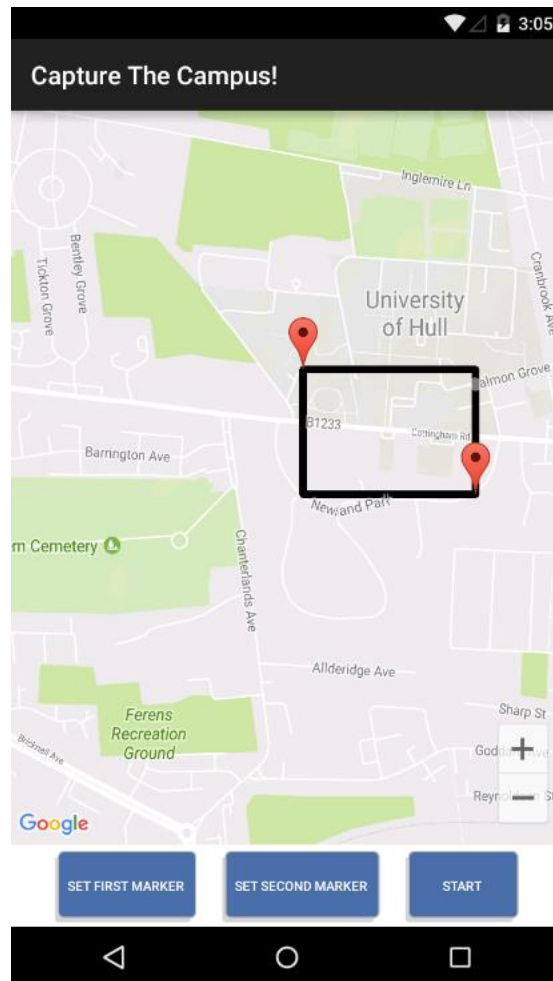


Figure 32: This image shows the user interface for the Set activity after the markers or flags have been set.

The image above (Figure 32) shows the final user interface for the Set activity after the markers or flags have been placed upon the map to define the initial play area. As can be seen the final user interface in this instance made a use of the markers and polygons, as discussed previously, to represent the markers that have been set to define the initial play area and to represent the play area itself.

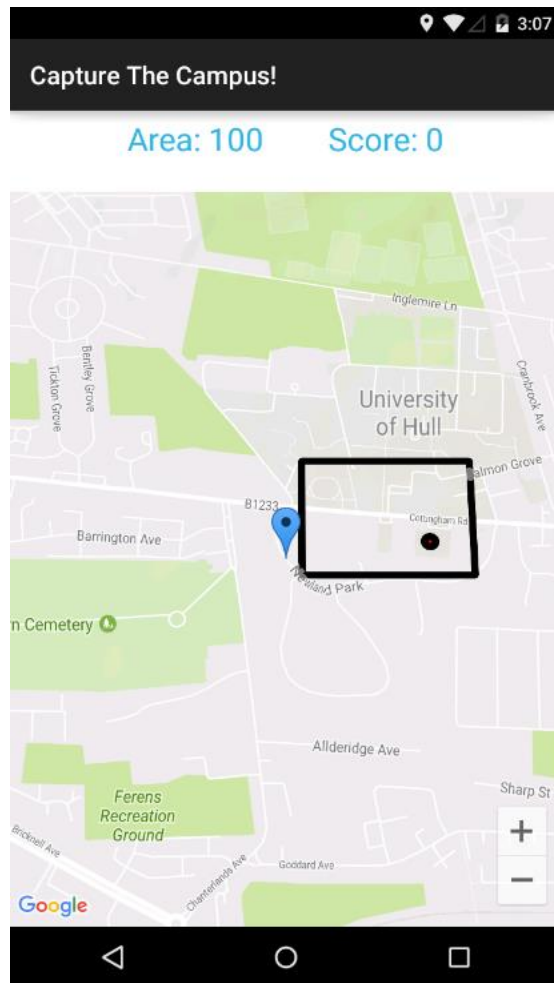


Figure 33: This image shows the user interface for the Game activity just after the game has been started.

The image above (Figure 33) shows the final user interface for the Game activity just after the game has started.

As can be seen the final user interface in this instance made a use of the markers and polygons, as discussed previously, to represent the player and to represent the play area and enemy.

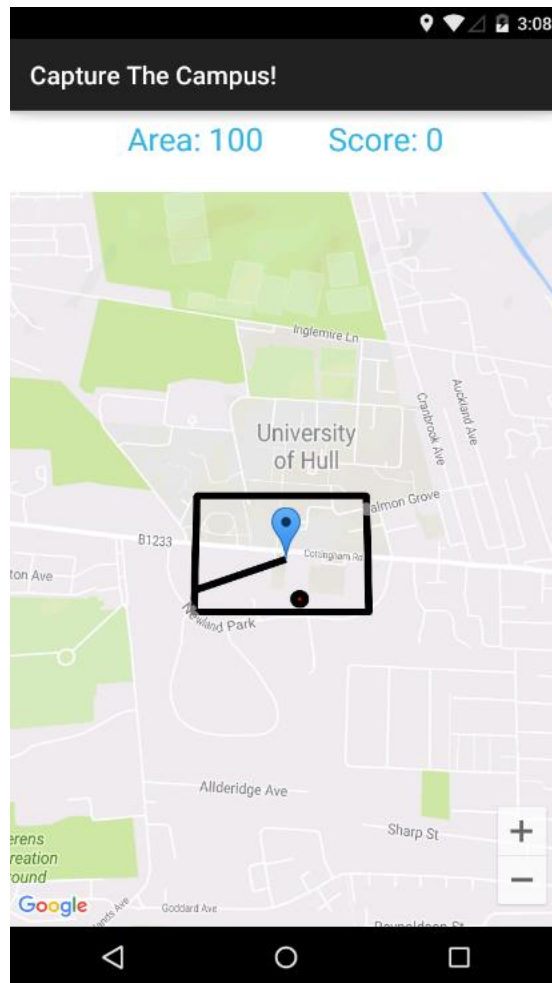


Figure 34: This image shows the user interface for the Game activity as a player is attempting to capture an area.

The image above (Figure 34) shows the final user interface for the Game activity as a player is attempting to pass through the play area bisecting it in two and capturing the smallest are for them self.

As can be seen the final user interface in this instance made a use of the polylines, as discussed previously, to represent the player and to represent the play area and enemy.

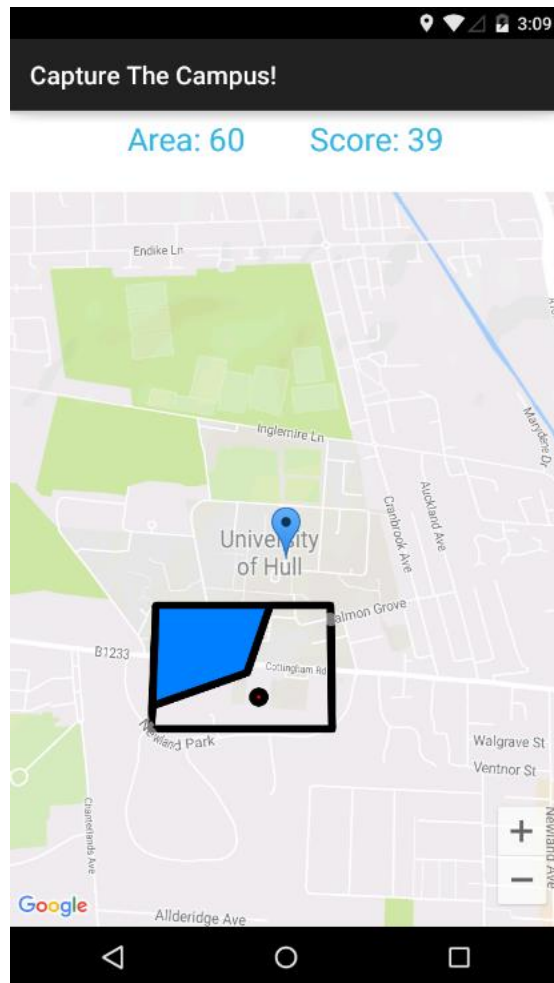


Figure 35: This image shows the user interface for the Game activity just after a player has captured an area.

The image above (Figure 35) shows the final user interface for the Game activity just after a player has managed to successfully capture an area of the play area. As can be seen the area that has been captured has been coloured in the same colour as the marker that represents the player and the size of the area clipped has been added to the player's score and removed from the overall area of the play area.

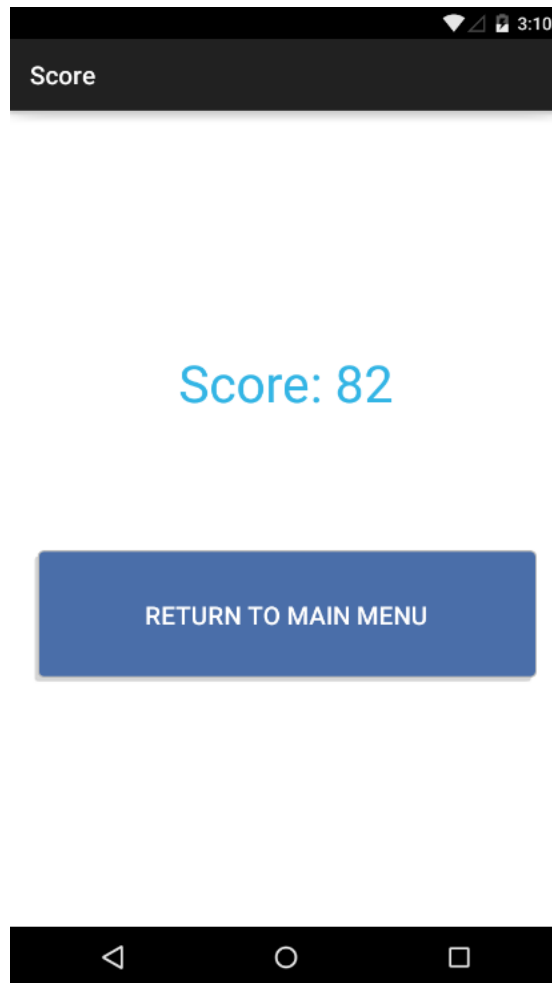


Figure 36: This image shows the user interface for the Score activity.

The image above (Figure 36) shows the final user interface for the Score activity. As can be seen the player in this instance finished the game with a score of 82, there is also a user interface option to return to the main menu.

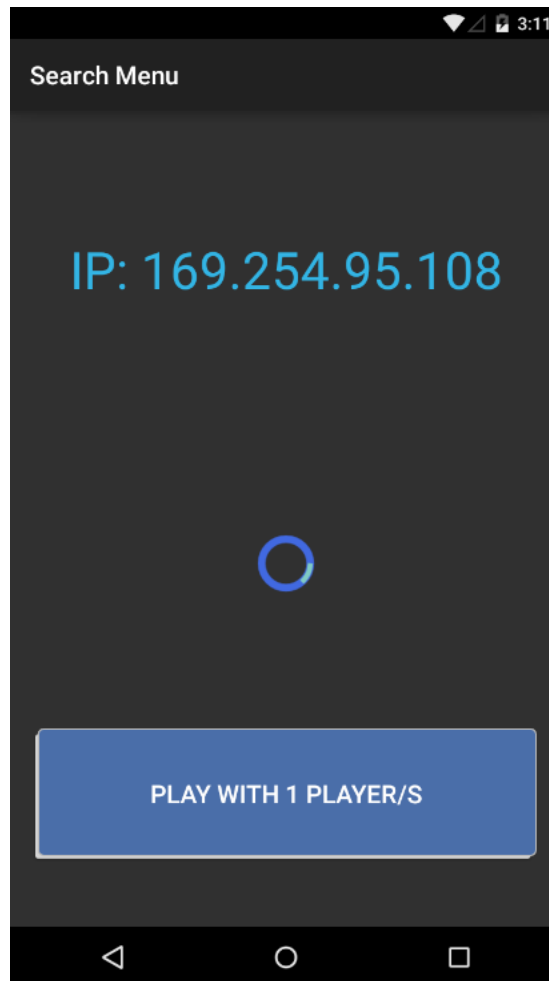


Figure 37: This image shows the user interface for the Host activity.

The image above (Figure 37) shows the final user interface for the Host activity. As can be seen in this instance the IP address of the host has been captured as 169.254.95.108, there is also a user interface option to start the game with the one player currently connected and synced with the host device.

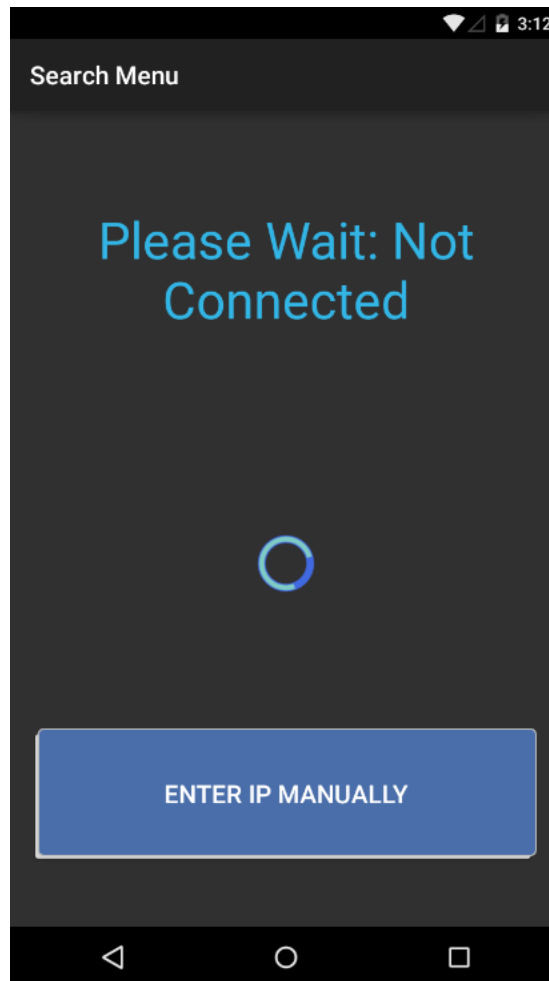


Figure 38: This image shows the user interface for the Join activity.

The image above (Figure 38) shows the final user interface for the Join activity. As can be seen in this instance the device has not yet managed to connect and synchronise with a host device, there is also a user interface option to manually enter the IP address of the host device.

4.3 System Implementation

4.3.1 Processes and Methodology

Software development methodologies are frameworks which are used to help to control the process of developing a large software development project using structure and planning.

When undertaking a large software development project, it is usually advantageous to select a software development methodology to follow, with the intent of better planning and time management (IT Knowledge portal, 2017).

The image below (Figure 39) shows a graphical representation of three more traditional software development methodologies which could have been used to aid in the development of this project; Waterfall, Prototyping and Spiral methodologies.

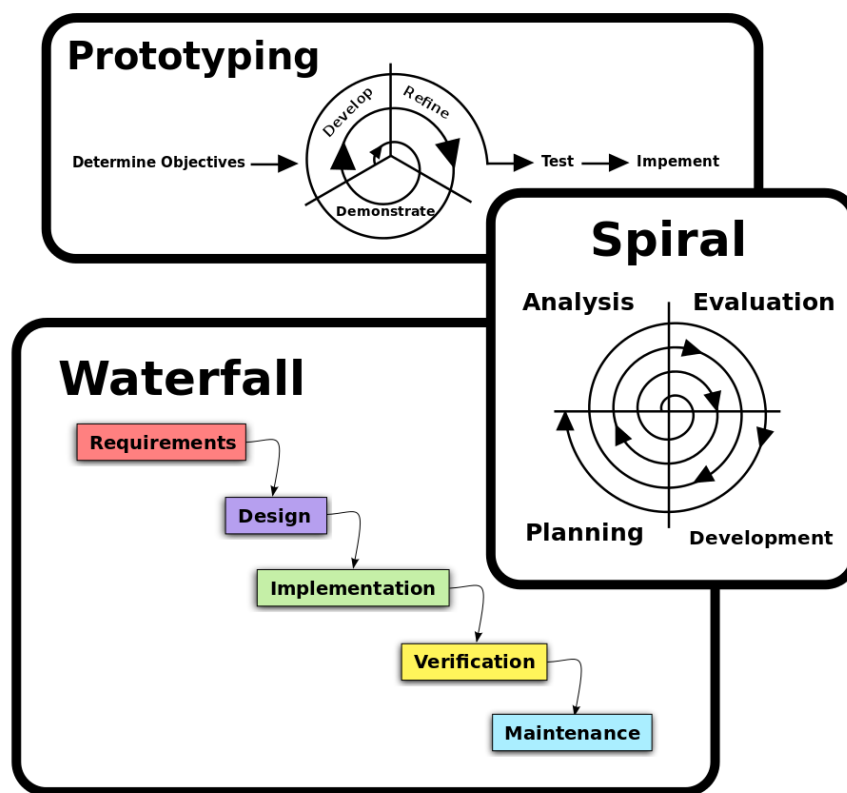


Figure 39: This image shows a collage of the three main software development models (Wikipedia, 2017).

The Waterfall methodology, which is also called the traditional methodology, is the oldest and most popular form of software development methodologies, it is a rather rigid and linear approach to software development where each phase is completed before the next with no skipping from one stage to another, tasks flow like a river or waterfall from one stage to another, usually from design through implementation to testing, hence the name (Howcroft & Carroll, 2000). This can clearly be seen in the bottom left hand corner of the diagram above (Figure 39).

The main advantage to the Waterfall methodology is that it is easy to understand and implement for managers in a business context because this form of project management can be applied to any project not necessarily specifically a software development project, thus it

is possible for a manager with no prior knowledge of computer science to manage a software development project in this manner. Plus because of the way the Waterfall methodologies aims are defined, in detail, the project should theoretically be delivered on time (Schwaber, 1997).

However, in practice because the Waterfall methodology does not consider the inevitable changes in the specification of a software development project, due to such factors as scope creep and because the Waterfall methodology does not allow or plan for changes to be made that were not thought of during the initial planning stage projects managed using the Waterfall methodology projects usually run overtime (Shellnut, et al., 1999).

In this instance, a software development methodology based loosely on both the Prototyping and Spiral methodologies was used called the Iterative and Incremental Development methodology.

This software development methodology as can be seen in the image below (Figure 40) fixes some of the issues experienced in the Waterfall methodology. Rather than development being a rigid linear process from start to end, certain goals are set and these goals are delegated out in cycles whereby at the end of each cycle there is a new piece of functionality.

The main advantage to the Iterative and Incremental Development methodology is that the product is more often closer in functionality to what was required than the Waterfall methodology because of the ability to continuously demonstrate new features and adjust based on feedback. Also, because of the continuous testing of functionality it is more likely that issues in the time plan for a project are identified and dealt with earlier in the lifetime of the project, meaning that often projects are more likely to be on time than with the Waterfall methodology (Larman & Basili, 2003).

One disadvantage of the Iterative and Incremental Development methodology is that because each developer or group of developers is given a task to complete from start to finish they must all be trained to a high level in all aspects of the development process (Farcic, 2014). However, because this project was managed and developed by one person this issue is moot.

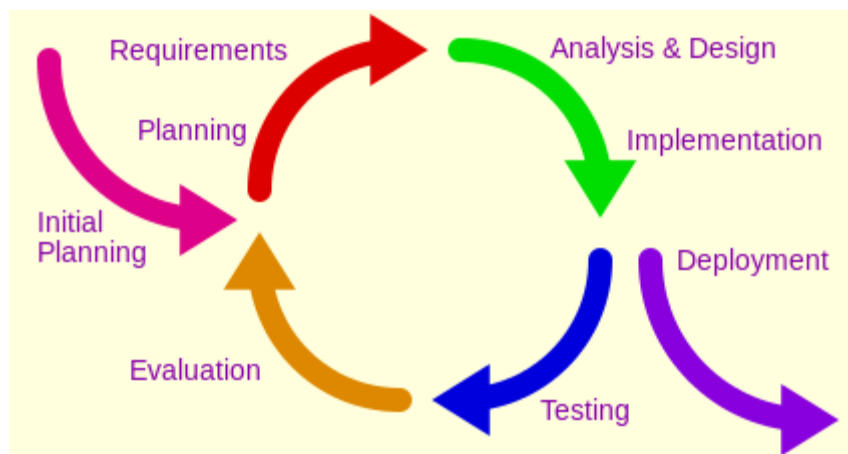


Figure 40: This image shows a graphical representation of the iterative and incremental development software development model (Wikipedia, 2017).

To save time and resources as well as to make the program more accessible and understandable the standard of self-documenting code was followed throughout this project (Schach, 2007).

4.3.2 Version Control

When undertaking a large software development project, it is usually advantageous to use a form of version control.

Version control allows a large team of developers to contribute simultaneously to the same code base without causing mass conflicts with different versions of the same software (Pilato, et al., 2002).

Version control also allows for the branching of a project if two different development teams want to take the project in different directions (Pilato, et al., 2002).

Version control can also save a project if something terminal was to occur to the currently checked out version of the repository. Such as, for the current version to become corrupted. Version control can solve this issue as most forms of version control allow for the project to be rolled back to any version of the project that is stored in commit history (Pilato, et al., 2002).

For this project Subversion was chosen as the main form of version control, this is because the University of Hull offers its own Subversion server. Although versions of the project were also stored on GitHub.

The image below (Figure 41) shows the commit history of the project.

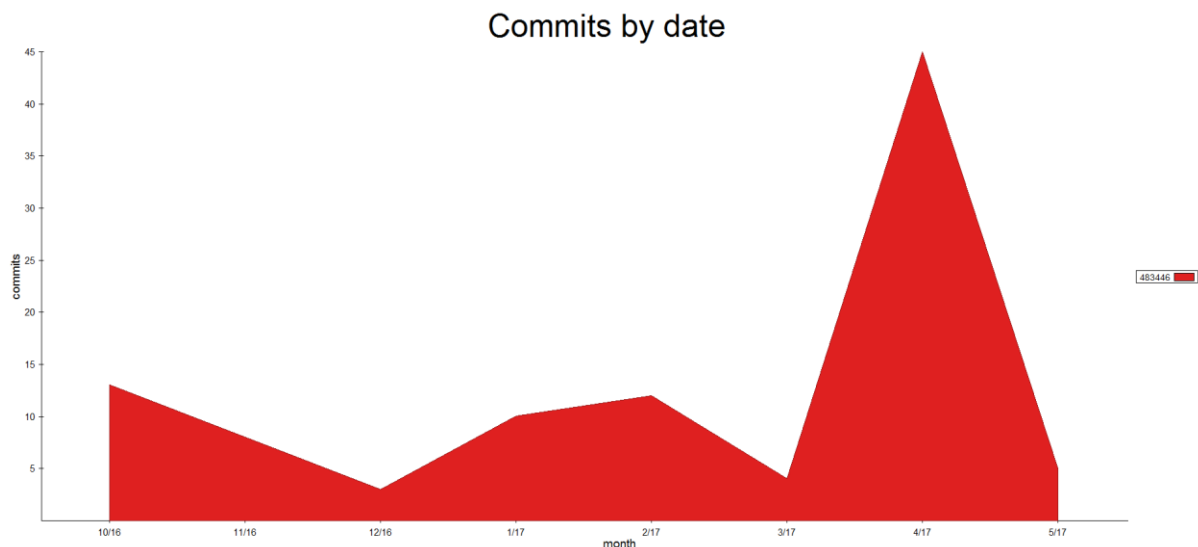


Figure 41: This image shows a graphical representation of the commit history for this project.

4.3.3 Polygon Clipping Algorithm

Temporary vertex = first Play area vertex

Do While (Temporary vertex != null)

If (first Path vertex intersects line (Temporary vertex, next Temporary vertex))

Add first Path vertex after Play area vertex at Temporary vertex

Else

Temporary vertex = next Temporary vertex

Loop

Temporary vertex = first play area vertex

Do While (Temporary vertex != null)

If (last Path vertex intersects line (Temporary vertex, next Temporary vertex))

Add last Path vertex after Play area vertex at Temporary vertex

Else

Temporary vertex = next Temporary vertex

Loop

Temporary vertices one = Path vertices

Temporary vertices two = Path vertices

Temporary vertex = Play area vertex at last Path vertex

Do While (Temporary vertex != Play area vertex at first Path vertex)

Temporary vertices one += Temporary vertex

Temporary vertex = next Temporary vertex

Loop

Temporary vertex = Play area vertex at last Path vertex

Do While (Temporary vertex != Play area vertex at first Path vertex)

Temporary vertices two += Temporary vertex

Temporary vertex = previous Temporary vertex

Loop

If (area of Temporary vertices one > area of Temporary vertices two)

Play area vertices = Temporary vertices one

Score += area of Temporary vertices two

Else

Play area vertices = Temporary vertices two

Score += area of Temporary vertices one

The above pseudo code shows an easily readable high level description of the general functionality of the algorithm to bisect the play area by the path of a player. The full code for this algorithm can be seen below (Appendix F:).

The play area and path are both represented by linked lists of vertices, if the linked list of vertices for the path could be inserted into the linked list of vertices for the play area then two temporary linked lists of vertices could be created which represent two polygons which share a common edge, which would be the path.

First, the point where the path initially intersects the play area needs to be found and the vertex representing this point needs to be inserted into the linked list of play area vertices. To do this, as can be seen in the pseudo code above, a temporary linked list node is created which points to the first vertex of the play area, then within a while loop each line segment of the play area, which is defined as the line segment represented by the temporary linked list node and the next node in the temporary linked list, is checked to see if it intersects with the first vertex of the path. If the first vertex of the path intersects with the line segment then the first vertex of the path is added to the play area after the node which is represented by the temporary linked list node.

This method is then repeated but rather than looking for an intersection between the line segment and the first vertex of the path an intersection is to be found between the line segments of the play area and the last vertex of the path.

Once the first and last vertices of the path have been inserted into the play area the two polygons which are formed when the play area is clipped by the path can be created. Firstly, the path can be added to the list of vertices for both polygons, this is because the two polygons share the edge represented by the path, then the clockwise polygon is created through the use of another temporary linked list node which is initialised at the point in the play area linked list where the last vertex of the path has been inserted, then each vertex in the linked list after this node is added to the first temporary polygon until the temporary linked list node equals the first vertex of the path.

This method is then repeated to create the anticlockwise polygon but rather than moving the temporary linked list node to the next node in the play area linked list and adding that node to the anticlockwise polygon on each iteration the temporary linked list node is initialised at the point in the play area linked list where the last vertex of the path has been inserted, then each vertex in the linked list before this node is added to the second temporary polygon until the temporary linked list node equals the first vertex of the path.

After the vertices for the first clockwise polygon and the vertices for the second anticlockwise polygon have been added to their respective polygon then the larger of the two polygons must be established. To do this a function has been created in the static Maths class to find the area of a polygon.

If the first polygon is larger than the second polygon then the play area vertices are set to the same linked list as the vertices of the first polygon and the score of the player is increased relative to the area of the second polygon. If the second polygon is larger than the first polygon then the play area vertices are set to the same linked list as the vertices of the second polygon and the score of the player is increased relative to the area of the first polygon.

4.3.4 Polygon Tessellation Algorithm

```
Do While (length of Vertices > 3)
    For (0 to length of Vertices)
        If ((Vertex at counter, Vertex at counter + 1, Vertex at counter + 2) create Triangle)
            Triangles += Triangle
            Vertices -= Vertex at counter + 1
    Loop
Loop
Triangles += (Vertex at 0, Vertex at 1, Vertex at 2)
```

The above pseudo code shows an easily readable high level description of the general functionality of the algorithm to tessellate the play area.

The full code for this algorithm can be seen below (Appendix G:).

To find a random point within the play area first the play area must be tessellated into regular triangles.

This algorithm functions almost entirely within a single while loop which exits when the number of vertices in the temporary list of vertices in the algorithm drops to three or below. The while loop contains a for loop which iterates over the contents of the list of vertices seeking three sequential vertices which form a triangle, which in the terms of a tessellation algorithm is usually called an ear.

An ear is defined as a triangle that is not concave, its line segments do not intersect with the line segments of any other triangle and it does not contain any other points of the larger polygon with itself.

When an ear is found that meets these requirements it is added to a list of triangles and its middle vertex is removed from the list of vertices this is because the middle vertex can be assumed to be part of no other triangle.

Once the number of vertices in the temporary list of vertices in the algorithm drops to three these three final vertices are added to the list of triangles as the final triangle, this list of triangles is then returned from the algorithm.

4.3.5 Wandering Algorithm

```
Do While (true)
    If (Position is not inside Play area)
        Position = random position inside Play area
        Degrees = random number between 0 and 360
    Else
        Previous position = Position
        Unit vector = Degrees to unit vector
        Position = Unit vector * Velocity of object
        Acceleration = random number between -45 and 45
        Degrees += Acceleration
        If (Degrees > 360)
            Degrees = Acceleration - (360 - Degrees)
        Else
            If (Degrees < 0)
                Degrees = 360 - (Acceleration - Degrees)
        If (Position collides with object)
            Position = Previous position
            Degrees = reflected Degrees
    Wait 1 second
Loop
```

The above pseudo code shows an easily readable high level description of the general functionality of the algorithm to calculate the enemies wandering behaviour. The full code for this algorithm can be seen below (Appendix H:).

The enemy is represented by a circle that changes size depending on the area of the initial play area, its behaviour runs on a separate thread to the rest of the game logic, this is because the enemy is required to move at set intervals consistently and as such its logic is contained within a while true loop.

Firstly, the location of the enemy is checked to see if it is inside the play area, if it is not then a new random location is acquired using the polygon tessellation algorithm above, then a new trajectory is gotten by using the inbuilt random method. This trajectory is represented using degrees between zero and three hundred and sixty, this is because it is a lot easier to work on a scale based on whole numbers which is linear and contains only one value, radians on the other hand would not have the granularity required and a unit vector would be much harder to keep consistent because of its two axis.

Next, the algorithm goes on to save the current position before it is changed, then the trajectory in degrees is converted into a unit vector as it is much easier to manipulate a position with a unit vector. The position of the enemy is then changed by the unit vector multiplied by its velocity.

The next part is the section that makes the enemy wander, a random acceleration between minus 45 and 45 is applied to the trajectory. This acceleration represents a cone of ninety degrees in front of the enemy into which it could move, this cone changes on each iteration of the wandering algorithm.

The only point which could not manipulate the path of the enemy would be if it were to collide with an object, if this were to happen the enemy is moved back to its previous position, to prevent it from becoming stuck on an object and its trajectory is reflected by the normal to the object that it collided with.

Finally, the thread waits for one second before starting again from the beginning.

4.4 Test Design

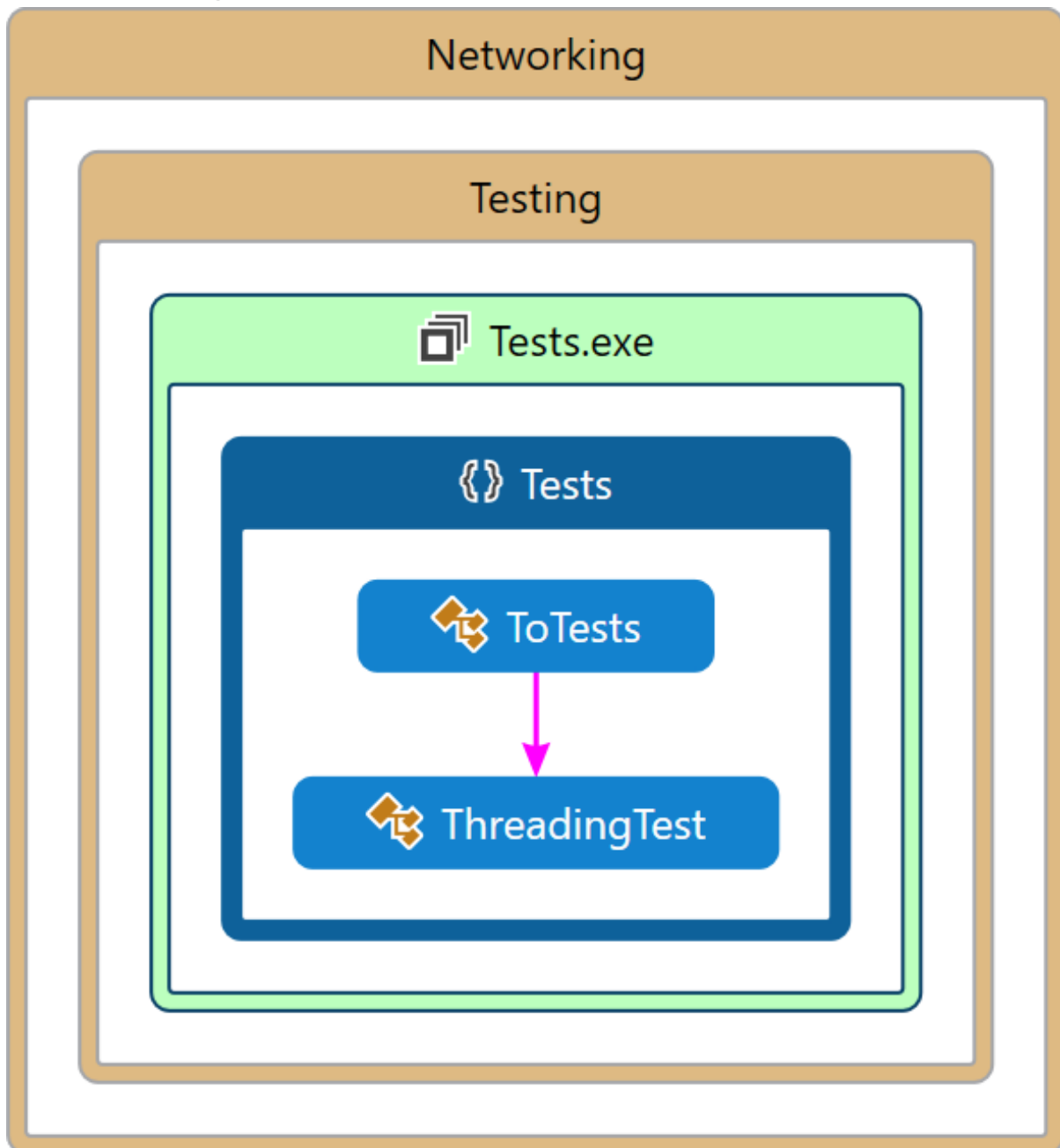


Figure 42: This image shows a diagram that describes the structure of the Tests subsystem by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Wikipedia, 2016).

The diagram above (Figure 42) shows a graphical representation of the solution for the tests, this diagram is called a code map.

This code map shows both the projects of the solution and classes contained within these projects which when compiled create the client.

Included in the code map are arrows the full legend for the code map can be seen below (Appendix B:).

As can be seen from the code map above (Figure 42) in the solution for the tests there is one project; an application called Tests contained within a folder named Testing which is itself contained within a folder called Networking.

The Tests application is a test testing application currently for the TCP server.

The Tests application contains two class; the first class allows the selection of the specific test to be applied, this class is called the ToTests class and the second class which is one of the tests which can be applied this class is called the ThreadingTest class.

The ThreadingTest class is used to test the TCP servers threading capabilities, this test works using a parallel for loop which on each iteration requests a piece of data from the server.

Most of the classes contained within the projects of this solution and their contents are private.

Being private allows for the application of encapsulation (Noble, 2002).

#	Test	Expected Result
1	Start Watchdog	Starts TCP server
2	Wait more than 10 seconds	Closes Watchdog and TCP server
3	Start Client	Starts Client
4	Start UDP server	Starts UDP server
5	Send UDP broadcast to Client	Receive UDP server IP and port
6	Close UDP server	Closes UDP server
7	Start TCP server	Starts TCP server
8	Send identifier to TCP server	Receive error message
9	Send identifier followed by resource to TCP server	Receive acknowledgement of message receipt message
10	Send identifier to TCP server	Receive resource
11	Close TCP server	Closes TCP server
12	Start TCP server	Starts TCP server
13	Send identifier to TCP server	Receive resource
14	Close TCP server	Closes TCP server
15	Close Client	Closes Client
16	Start Tests	Starts Tests
17	Start threading test	Starts TCP server
18	Test 10 concurrent threads	TCP server accepts 10 concurrent messages
19	Test 20 concurrent threads	TCP server accepts 20 concurrent messages
20	Test 30 concurrent threads	TCP server accepts 30 concurrent messages
21	Test 100 concurrent threads	TCP server accepts 100 concurrent messages
22	Start Capture the Campus!	Starts Capture the Campus!
23	Select single player game	Starts Set activity and camera moves to current location
24	Select back	Returns to Menu activity
25	Select single player game	Starts Set activity and camera moves to current location
26	Select start game	Game does not start
27	Select single player game	Starts Set activity and camera moves to current location
28	Select select first flag	Able to place first flag on map

29	Select a location on the map	The first flag is placed at that location on the map
30	Select another location on the map	The first flag is placed at that location on the map
31	Select start game	Game does not start
32	Select select second flag	Able to place second flag on map
33	Select a location on the map	The second flag is placed at that location on the map
34	Select another location on the map	The second flag is placed at that location on the map
35	Select start game	Starts Game activity with the play area defined by the flags and camera moves to current location
36	Rotate device	Map rotates with device
37	Select back	Returns to Menu activity
38	Select single player game	Starts Set activity and camera moves to current location
39	Select select first flag	Able to place first flag on map
40	Select a location on the map	The first flag is placed at that location on the map
41	Select select second flag	Able to place second flag on map
42	Select a location on the map	The second flag is placed at that location on the map
43	Move outside of play area	Nothing
44	Select start game	Starts Game activity with the play area defined by the flags and camera moves to current location
45	Move inside play area	Path is drawn from the point of entrance into the play area to the current location
46	Select back	Returns to Menu activity
47	Select single player game	Starts Set activity and camera moves to current location
48	Select select first flag	Able to place first flag on map
49	Select a location on the map	The first flag is placed at that location on the map
50	Select select second flag	Able to place second flag on map
51	Select a location on the map	The second flag is placed at that location on the map
52	Move inside play area	Nothing
53	Select start game	Starts Game activity with the play area defined by the flags and camera moves to current location
54	Move outside of play area	Nothing
55	Move inside play area	Path is drawn from the point of entrance into the play area to the current location
56	Move outside play area via the same line segment as which was entered	Path is drawn from previous position to the point of exit and the smallest area is captured, added to the score and removed from the total area of the play area
57	Move into captured area	Path is not drawn

58	Move inside play area	Path is drawn from the point of entrance into the play area to the current location
59	Move outside play area via an opposing line segment as which was entered	Path is drawn from previous position to the point of exit and the smallest area is captured, added to the score and removed from the total area of the play area
60	Move inside play area	Path is drawn from the point of entrance into the play area to the current location
61	Allow enemy to intersect path	Path is removed and score is halved
62	Move outside of play area	Nothing
63	Pass through the play area ensuring that the enemy is in the area captured	The enemy repositions itself at a random location within the new play area
64	Pass through the play area until 75% of the play area has been captured	Starts Score activity with correct score displayed
65	Select return to menu	Returns to Menu activity
66	Start Capture the Campus! on another device	Starts Capture the Campus! on another device
67	Select host multiplayer game on device 1	Starts Host activity on device 1
68	Select join multiplayer game on device 2	Starts Join activity on device 2 and a UDP broadcast is picked up from device 1, device 2 connects to device 1 and the messages displayed on both devices are updated to represent this
69	Select start game on device 1	Starts Set activity on device 1 nothing happens on device 2
70	Select select first flag on device 1	Able to place first flag on map on device 1
71	Select a location on the map on device 1	The first flag is placed at that location on the map on device 1
72	Select select second flag on device 1	Able to place second flag on map on device 1
73	Select a location on the map on device 1	The second flag is placed at that location on the map on device 1
74	Select start game on device 1	Starts Game activity with the play area defined by the flags and camera moves to current location on both devices
75	Move device 1 inside play area	Path is drawn from the point of entrance of device 1 into the play area to the current location of device 1 on both devices
76	Move device 1 outside of play area	Path is drawn from previous position of device 1 to the point of exit of device 1 and the smallest area is captured, added to the score of device 1 and removed from the total area of the play area on both devices
77	Pass both devices through the play area so that their paths intersect	Path is removed and score is halved on both devices
78	Pass through the play area on both devices until 75% of the play area has been captured	Starts Score activity with correct score displayed on both devices
79	Close Capture the Campus!	Closes Capture the Campus!

Tests 1-2 deal with testing the functionality of the Watchdog, including; starting up and shutting down both the UDP and TCP servers.

Test 3 starts the debugging client.

Tests 4-6 deal with testing the functionality of the UDP server, including; sending and receiving UDP broadcasts.

Tests 7-14 deal with testing the functionality of the TCP server, including; throwing errors, recording usernames and locations, recalling a given location for a given username and rebuilding the database after shutting down.

Test 15 shuts the debugging client down.

Tests 16-21 deal with using the Tests application to test the functionality of the TCP server specifically its capability to deal with multiple concurrent users

Tests 22-25 deal with testing the functionality of the Menu activity in the Capture the Campus! application, including; its functionality as a menu to select the different game modes and the ability to return to the menu.

Tests 26-35 deal with testing the functionality of the Set activity in the Capture the Campus! application, including; the ability to set and move flags.

Tests 36-60 deal with testing the functionality of the Game activity in the Capture the Campus! application, including; the ability to rotate the map, the ability to pass through the play area and capture area which then affects the score etc. and if there are any bugs in the area capturing algorithms.

Tests 61-65 deal with testing the functionality of the enemy in the Game activity in the Capture the Campus! application, including; how the enemy spawns and respawns after being removed from the play area and how the enemy interacts with the players.

Tests 64-65 deal with testing the functionality of the Score activity in the Capture the Campus! application, including; if the Score activity is called at the correct time with the correct score.

Tests 66-79 deal with testing the functionality of the multiplayer aspects in the Capture the Campus! application, including; if the devices can make a connection with each other, if the device which is hosting the multiplayer game can synchronise the game state with each client device and if the multiplayer game functionality works.

5 Evaluation

5.1 Project Achievements

Generally, progress on the project was impeded by multiple factors:

Firstly, the main computers that were chosen to be used to complete the project did not in fact have the required development software installed to develop the project. In addition, the development software was never installed by the administrators of the computers, plus the administrators of the computers never allowed administrative access to the computers to allow the installation of the development software.

Furthermore, the backup computer-that was to be used in situations like this-that has the development software installed-does not have the required specifications to run the development software to an adequate performance level. This means that anything that required the development software did not begin development until late into the project. However, this does mean that the extra time was available to fully optimise and add additional features-that would not have otherwise been added-to the areas of the project that could be developed without this development software.

Secondly, the initial time plan (Appendix J:) did not consider other extraneous commitments when allotting time to the project. Time should have been allowed to complete other compulsory projects on time, as such it appears that the project was behind schedule from the start when in fact the initial time plan was inherently flawed and unachievable.

Thus, a new task list (Appendix K:) and time plan (Appendix L:) was created considering the issues mentioned above:

The tasks that require the specific development software mentioned above were delayed until after Christmas, this gave enough time for a solution to the problem to present itself, a virtual machine was acquired which could be used to develop the project on.

Sections of time were blocked out for the completion of extraneous compulsory commitments, this included the reworking of the remaining time allocated to the project to ensure the smoothest development experience possible.

The addition of a multigame server was abandoned around Christmas as the scope of the project had to be reduced because of the issues mentioned above.

Because of the delays caused by the lack of development software sections had to be amended to the risk analysis (Appendix M:) section to reflect any risks that had been encountered that were not originally planned for. Including; a section detailing the risk of missing development software, a section detailing the risk of the backup computer being insufficient to develop on and a section detailing the risk of insufficient funds to upgrade the backup computer.

Because of the nature of the project there were practically no ethical concerns. There was no intention to conduct any kind of experiment so ethical approval was not needed and any other ethical concerns, for example, use of the project ending in the injury of members of the public had been covered in the risk analysis section (Appendix M:) or were beyond the scope of the project.

A new task list (5.1.1) and time plan (5.1.2) was created to reflect the actual progress throughout the project.

5.1.1 Final Task List

#	Task Name	Description	Duration (weeks)
1	Research	Conduct research that will aid in the writing of the report and design of the project	10
2	Initial report	Write the initial report deliverable	2
3	Create server client library and test wrapper	Create a client library that is capable of interfacing with the server and a test wrapper which will allow the client to be used as both a library and an application	5
4	Create TCP server and test wrapper	Create a TCP server that can accept TCP packets from the client and a test wrapper which will allow the TCP server to be used as both a library and an application	2
5	Create UDP server and test wrapper	Create a UDP server that can send UDP packets to the client which will be used to identify the IP of the TCP server and a test wrapper which will allow the UDP server to be used as both a library and an application	1
6	Add multithreading capabilities to TCP server	Add multithreading to the TCP server so that it is capable of accepting multiple client request simultaneously	2
7	Add logging capabilities to the TCP server	Add logging to the TCP server so that it is capable of recovering from a fatal error. The log can be used to debug the TCP server.	2
8	Work on other projects	Time set aside to work on other projects	9
9	Interim report	Write the interim report deliverable	3
10	Create main menu for game	Create a main menu for the game that will be displayed when the game is started and between every game instance. The main menu should display all options for game types and settings	1
11	Add map to game activity	Add a suitable map to the game screen	2

12	Add player character to game screen	Add a representation of the player's position to the game screen	2
13	Move player character and camera with player's movements	Move the player character to the latitude and longitude coordinate given by the device's GPS sensor at a reasonable interval. The camera should also be moved to the player's location	2
14	Add method to define the play area	Add a method that allows the player to define the initial play area. This should work by allowing the placement of two markers representing two opposing sides of a rectangular play area.	2
15	Add a method to track the player's path and display it on the map	Add a method which creates a polyline based on the latitude and longitude coordinates of the player's movements	2
16	Add a method to determine if an object is inside a polygon	Add a method that returns if a point is within the play area. This should work for any play area.	2
17	Add a method to detect an intersection between a sphere and a line segment	Add a method which finds the point where two line segments intersect if they intersect	2
18	Add a method to clip a line segment by another line segment	Add a method which removes the part of a line segment which is beyond the point where it intersects another line segment	2

19	Add a method to clip a polygon by a line segment	Add a method which creates two polygons from the points where a line segment bisects a larger polygon	3
20	Add a method to find the larger of two areas	Add a method which calculates the area of and then determines the larger of two polygons	1
21	Add a method to redefine the play area	Add a method that redefines the play area as the larger of two polygons. The smaller polygon should be added to the map filled in with a relevant colour	1
22	Add a method to update the score and area	Add a method that updates the score with the area of a polygon. The area of the play area should also be updated with this area	1
23	Create a game over screen	Create a game over screen for the game that will be displayed when the game ends. The game over screen should display the players score for the previous game and return to the menu screen	1
24	Add enemy to game screen	Add a representation of the enemy to the game screen	1
25	Add a method to find a random point in a polygon	Add a method which returns a uniformly random latitude and longitude coordinate located within the play area	2
26	Add a method to find a random unit vector in a given range	Add a method which when given a unit vector returns a new unit vector randomly within a reasonable range of the input unit vector	2
27	Add a method to detect an intersection between a	Add a method that calculates the distance between a circle and a line segment. If the distance is found to be less than the radius of the circle then the method should return that they have intersected	2

	sphere and a line segment		
28	Add a method to calculate a reflected unit vector from a line segment	Add a method which given a unit vector and a line segment, calculates the normal to the line segment and then multiplies the unit vector by this normal giving a unit vector reflected from the line segment	2
29	Add a method to remove a path from the play area	Add a method which removes a path from the map on the game screen	2
30	Create a lobby for local multiplayer	Create a lobby for the local multiplayer game that will be displayed when the game is starting a local multiplayer game.	2
31	Add a method to find the IP address of a device	Add a method which finds the IP address of the host device	2
32	Add a method to connect two devices via a UDP server	Add a method that broadcasts the host's IP address between the host and clients devices using the UDP server	2
33	Add a method to synchronise two devices via a TCP server	Add a method that mediates communication between the host and clients devices using the TCP server. Information that should be synchronised includes; the initial play area, the position of the enemy and the current location of each player	2
34	Add a method to allow the game to update multiple	Add a method that is used to augment the game activity to update the information of multiple players simultaneously from the TCP server. This will be used to implement a local multiplayer game.	2

	players simultaneously		
35	Create a video demonstration of the project	Film a short video demonstration of the project	2
36	Final report	Write the final report deliverable	3

5.1.2 Final Time Plan

#	Task Name	University Calendar Weeks																																			
		4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36			
1	Research																																				
2	Initial report				D																																
3	Create server client library and test wrapper																																				
4	Create TCP server and test wrapper																																				
5	Create UDP server and test wrapper																																				
6	Add multithreading capabilities to TCP server																																				
7	Add logging capabilities to the TCP server																																				
8	Work on other projects																																				
9	Interim report																																				
10	Create main menu for game																																				
11	Add map to game activity																																				

5.1.3 Test Evaluation

#	Test	Expected Result	Results
1	Start Watchdog	Starts TCP server	Pass
2	Wait more than 10 seconds	Closes Watchdog and TCP server	Pass
3	Start Client	Starts Client	Pass
4	Start UDP server	Starts UDP server	Pass
5	Send UDP broadcast to Client	Receive UDP server IP and port	Pass
6	Close UDP server	Closes UDP server	Pass
7	Start TCP server	Starts TCP server	Pass
8	Send identifier to TCP server	Receive error message	Pass
9	Send identifier followed by resource to TCP server	Receive acknowledgement of message receipt message	Pass
10	Send identifier to TCP server	Receive resource	Pass
11	Close TCP server	Closes TCP server	Pass
12	Start TCP server	Starts TCP server	Pass
13	Send identifier to TCP server	Receive resource	Fail
14	Close TCP server	Closes TCP server	Pass
15	Close Client	Closes Client	Pass
16	Start Tests	Starts Tests	Pass
17	Start threading test	Starts TCP server	Pass
18	Test 10 concurrent threads	TCP server accepts 10 concurrent messages	Pass
19	Test 20 concurrent threads	TCP server accepts 20 concurrent messages	Pass
20	Test 30 concurrent threads	TCP server accepts 30 concurrent messages	Pass
21	Test 100 concurrent threads	TCP server accepts 100 concurrent messages	Pass
22	Start Capture the Campus!	Starts Capture the Campus!	Pass
23	Select single player game	Starts Set activity and camera moves to current location	Pass
24	Select back	Returns to Menu activity	Fail
25	Select single player game	Starts Set activity and camera moves to current location	Pass
26	Select start game	Game does not start	Pass
27	Select single player game	Starts Set activity and camera moves to current location	Pass
28	Select select first flag	Able to place first flag on map	Pass
29	Select a location on the map	The first flag is placed at that location on the map	Pass
30	Select another location on the map	The first flag is placed at that location on the map	Pass
31	Select start game	Game does not start	Pass

32	Select select second flag	Able to place second flag on map	Pass
33	Select a location on the map	The second flag is placed at that location on the map	Pass
34	Select another location on the map	The second flag is placed at that location on the map	Pass
35	Select start game	Starts Game activity with the play area defined by the flags and camera moves to current location	Pass
36	Rotate device	Map rotates with device	Fail
37	Select back	Returns to Menu activity	Fail
38	Select single player game	Starts Set activity and camera moves to current location	Pass
39	Select select first flag	Able to place first flag on map	Pass
40	Select a location on the map	The first flag is placed at that location on the map	Pass
41	Select select second flag	Able to place second flag on map	Pass
42	Select a location on the map	The second flag is placed at that location on the map	Pass
43	Move outside of play area	Nothing	Pass
44	Select start game	Starts Game activity with the play area defined by the flags and camera moves to current location	Pass
45	Move inside play area	Path is drawn from the point of entrance into the play area to the current location	Pass
46	Select back	Returns to Menu activity	Fail
47	Select single player game	Starts Set activity and camera moves to current location	Pass
48	Select select first flag	Able to place first flag on map	Pass
49	Select a location on the map	The first flag is placed at that location on the map	Pass
50	Select select second flag	Able to place second flag on map	Pass
51	Select a location on the map	The second flag is placed at that location on the map	Pass
52	Move inside play area	Nothing	Pass
53	Select start game	Starts Game activity with the play area defined by the flags and camera moves to current location	Pass
54	Move outside of play area	Nothing	Pass
55	Move inside play area	Path is drawn from the point of entrance into the play area to the current location	Pass
56	Move outside play area via the same line	Path is drawn from previous position to the point of exit and	Pass

	segment as which was entered	the smallest area is captured, added to the score and removed from the total area of the play area	
57	Move into captured area	Path is not drawn	Pass
58	Move inside play area	Path is drawn from the point of entrance into the play area to the current location	Pass
59	Move outside play area via an opposing line segment as which was entered	Path is drawn from previous position to the point of exit and the smallest area is captured, added to the score and removed from the total area of the play area	Pass
60	Move inside play area	Path is drawn from the point of entrance into the play area to the current location	Pass
61	Allow enemy to intersect path	Path is removed and score is halved	Pass
62	Move outside of play area	Nothing	Pass
63	Pass through the play area ensuring that the enemy is in the area captured	The enemy repositions itself at a random location within the new play area	Pass
64	Pass through the play area until 75% of the play area has been captured	Starts Score activity with correct score displayed	Pass
65	Select return to menu	Returns to Menu activity	Fail
66	Start Capture the Campus! on another device	Starts Capture the Campus! on another device	Pass
67	Select host multiplayer game on device 1	Starts Host activity on device 1	Pass
68	Select join multiplayer game on device 2	Starts Join activity on device 2 and a UDP broadcast is picked up from device 1, device 2 connects to device 1 and the messages displayed on both devices are updated to represent this	Pass
69	Select start game on device 1	Starts Set activity on device 1 nothing happens on device 2	Pass
70	Select select first flag on device 1	Able to place first flag on map on device 1	Pass
71	Select a location on the map on device 1	The first flag is placed at that location on the map on device 1	Pass
72	Select select second flag on device 1	Able to place second flag on map on device 1	Pass

73	Select a location on the map on device 1	The second flag is placed at that location on the map on device 1	Pass
74	Select start game on device 1	Starts Game activity with the play area defined by the flags and camera moves to current location on both devices	Pass
75	Move device 1 inside play area	Path is drawn from the point of entrance of device 1 into the play area to the current location of device 1 on both devices	Pass
76	Move device 1 outside of play area	Path is drawn from previous position of device 1 to the point of exit of device 1 and the smallest area is captured, added to the score of device 1 and removed from the total area of the play area on both devices	Pass
77	Pass both devices through the play area so that their paths intersect	Path is removed and score is halved on both devices	Pass
78	Pass through the play area on both devices until 75% of the play area has been captured	Starts Score activity with correct score displayed on both devices	Pass
79	Close Capture the Campus!	Closes Capture the Campus!	Pass

5.2 Further Work

As can be seen from the diagrams above there are several issues with the project that could still be addressed.

For starters, the rotation feature of the game activity does not work, there is barebones code written into the project but more time will have to be dedicated to get the functionality to work optimally.

Secondly, a bug was discovered where when returning to the main menu via the back arrow on the bottom left of the device the event handlers for the menu items were not re-established correctly, thus when selecting a menu item for the second time two instances of the item would be created rather than one and on the third attempt three would be created. Eventually so many would be created that it would crash the device. To get around this issue after selecting a menu item the menu is destroyed like any other activity and recreated after the game has ended.

Ideally this bug should be fixed as otherwise to get back to the menu after selecting a menu item the application needs to be restarted.

Also, some functionality had to be cut from the final application due to time constraints, this can be seen in the scope shift from task list to task list.

It would be nice to return to the project and implement some of this functionality. Including; the different multiplayer game modes such as team based multiplayer game modes and the option to play on a central server rather than one device being the host.

Additionally, because of the way that the mobile device handles its file directories the logging that was added to the TCP server was found to be none functional. More time needs to be invested into this area of the application as logging for the TCP server could be considered an integral part of the networking features and safeguards.

Finally, as the project stands there will be some issues with the network connectivity when deployed to real devices rather than emulators. When a device moves from access point to access point it is given a new IP address and as it stands there is no way to inform client devices if the host device acquires a new IP address. However, the project would still work if the host device stays on the same access point.

Some ways to fix this issue could include; using a middle man to pass the new IP address around or using a centralised server rather than a host architecture.

6 Conclusion

The project in its current state has met most if not all of the objectives that were initially set out for it. The software has had most of the functionality that it depends on implemented and it has been implemented to a decent standard with a suitable user interface.

There are some known bugs in the application, however, these bugs are not game breaking and with a short amount of additional work could be remedied.

If the further work stated previously was carried out this application would be a contender on the market place as it fulfils a niche that no other application currently fills.

Appendix A: Capture the Campus!

Capture the flag is a well-known game (sub-) genre that requires players to capture the flag. Often there are two opposing teams each of which has a flag that must be defended from the other team.

http://en.wikipedia.org/wiki/Capture_the_flag

This project requires the student to design and develop an augmented reality game for a GPS-enabled mobile device (preferably Windows Phone 7). The details of the gameplay are open to negotiation, but a suggestion is that several flags (or capture points) are distributed in GPS locations around the campus. The players are required to visit the location for a specified period to capture the location. The players accumulate score based on number of capture points held and time they are uncontested for. The status of the capture points should be persistent, with the game's progress being able to be tracked over multiple (perhaps unlimited) days.

The project involves databases for storing flag locations and capture logs etc.

Project Code: DJP3

Appendix B: Code Map Legend









	Inherits From
	Implements
	Calls
	Function Pointer
	Field Read
	Field Write
	Project Reference
	External Reference

Figure 43: This image shows the legend or key for the code map diagrams.

Appendix C: Code Map

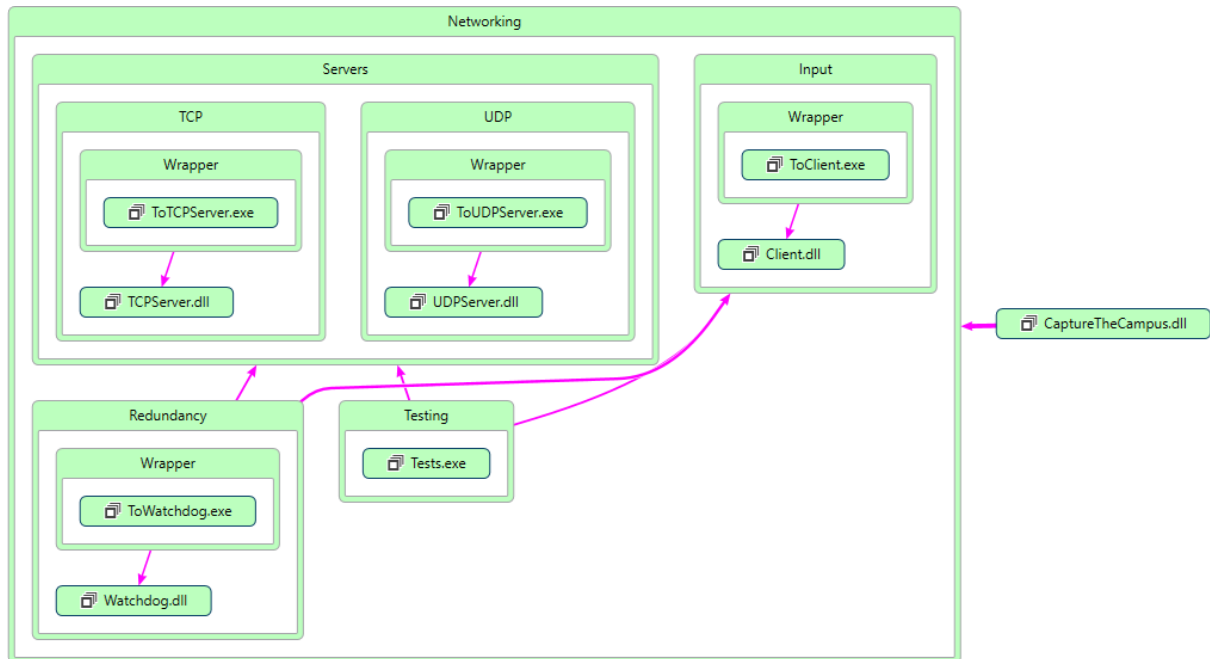


Figure 44: This image shows a diagram that describes the structure of the system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Wikipedia, 2016).

Appendix D: Networking Code Map

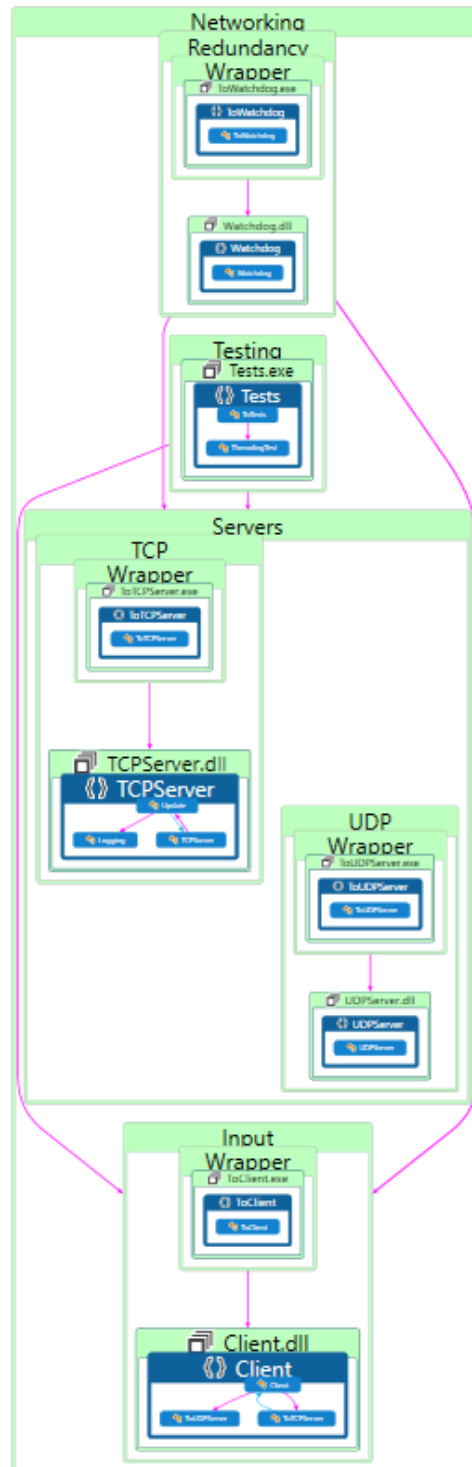


Figure 45: This image shows a diagram that describes the structure of the Networking subsystem by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Wikipedia, 2016).

Appendix E: CaptureTheCampus Code Map

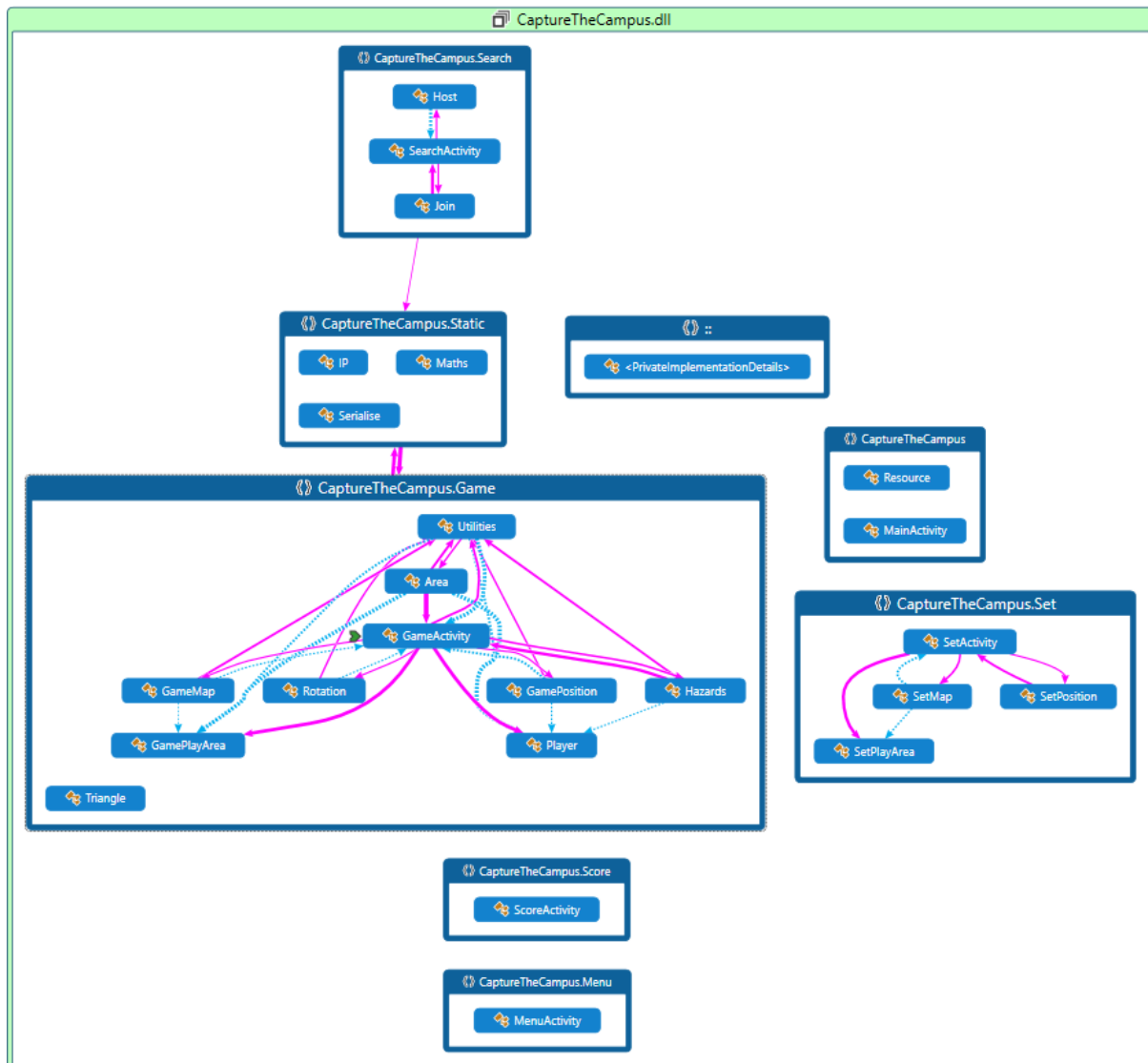


Figure 46: This image shows a diagram that describes the structure of the CaptureTheCampus subsystem by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Wikipedia, 2016).

Appendix F: Polygon Clipping Code

```
private void BuildArea(int playerPosition)
{
    CheckFirstPlayAreaIntersection(playerPosition);
}

private void CheckFirstPlayAreaIntersection(int playerPosition)
{
    if(CheckPlayAreaIntersection(gameActivity.playerArray[playerPosition].vertices.
First.Value))
    {
        CheckSecondPlayAreaIntersection(playerPosition);
    }
    else
    {
        gameActivity.finishBool = true;
    }
}

private void CheckSecondPlayAreaIntersection(int playerPosition)
{
    if(CheckPlayAreaIntersection(gameActivity.playerArray[playerPosition].vertices.
Last.Value))
    {
        BuildAreas(playerPosition);
    }
    else
    {
        gameActivity.finishBool = true;
    }
}
```

```

private bool CheckPlayAreaIntersection(LatLng value)
{
    LatLng firstExtendedPosition = Static.Maths.ExtendLineSegment(gameActivity,
value, Static.Maths.FindCentroid(gameActivity));

    LatLng secondExtendedPosition = Static.Maths.ExtendLineSegment(gameActivity,
value, firstExtendedPosition);

    gameActivity.gamePlayArea.verticesNode =
gameActivity.gamePlayArea.vertices.First.Next;

    while(true)
    {
        if(Static.Maths.DoIntersect(firstExtendedPosition,
secondExtendedPosition, gameActivity.gamePlayArea.verticesNode.Previous.Value,
gameActivity.gamePlayArea.verticesNode.Value))
        {
            gameActivity.gamePlayArea.vertices.AddBefore(gameActivity.gamePla
yArea.verticesNode, value);

            return true;
        }

        if(gameActivity.gamePlayArea.verticesNode.Next != null)
        {
            gameActivity.gamePlayArea.verticesNode =
gameActivity.gamePlayArea.verticesNode.Next;
        }
        else
        {
            if(CheckPlayAreaIntersectionCircularly(firstExtendedPosition,
secondExtendedPosition, value))
            {
                return true;
            }
            else
            {
                return false;
            }
        }
    }
}

```

```

private bool CheckPlayAreaIntersectionCircularly(LatLng firstExtendedPosition, LatLng
secondExtendedPosition, LatLng value)
{
    if(Static.Maths.DoIntersect(firstExtendedPosition, secondExtendedPosition,
gameActivity.gamePlayArea.vertices.First.Value,
gameActivity.gamePlayArea.vertices.Last.Value))
    {
        gameActivity.gamePlayArea.vertices.AddLast(value);

        return true;
    }
    else
    {
        Log.Error("CheckPlayAreaIntersectionCircularly", "Error in play area");

        return false;
    }
}

private void BuildAreas(int playerPosition)
{
    LinkedList<LatLng> firstPolygonVertices = new LinkedList<LatLng>();
    LinkedList<LatLng> secondPolygonVertices = new LinkedList<LatLng>();
    AddPath(playerPosition, firstPolygonVertices, secondPolygonVertices);
    AddFirstPolygon(playerPosition, firstPolygonVertices);
    utilities.SetPolygon(firstPolygonVertices);
    AddSecondPolygon(playerPosition, secondPolygonVertices);
    utilities.SetPolygon(secondPolygonVertices);
    TestAreas(playerPosition);
}

```

```

private void AddPath(int playerPosition, LinkedList<LatLng> firstPolygonVertices,
LinkedList<LatLng> secondPolygonVertices)
{
    gameActivity.playerArray[playerPosition].verticesNode =
gameActivity.playerArray[playerPosition].vertices.First;

    while(true)
    {
        firstPolygonVertices.AddLast(gameActivity.playerArray[playerPosition].verticesNode.Value);

        secondPolygonVertices.AddLast(gameActivity.playerArray[playerPosition].verticesNode.Value);

        if(gameActivity.playerArray[playerPosition].verticesNode.Next != null)
        {
            gameActivity.playerArray[playerPosition].verticesNode =
gameActivity.playerArray[playerPosition].verticesNode.Next;
        }
        else
        {
            break;
        }
    }
}

```

```

private void AddFirstPolygon(int playerPosition, LinkedList<LatLng>
firstPolygonVertices)
{
    if(gameActivity.gamePlayArea.vertices.Find(gameActivity.playerArray[playerPosit
ion].vertices.Last.Value).Next != null)
    {
        gameActivity.gamePlayArea.verticesNode =
gameActivity.gamePlayArea.vertices.Find(gameActivity.playerArray[playerPosition
].vertices.Last.Value).Next;
    }
    else
    {
        gameActivity.gamePlayArea.verticesNode =
gameActivity.gamePlayArea.vertices.First;
    }

    while(true)
    {
        if(gameActivity.gamePlayArea.verticesNode.Value !=
gameActivity.playerArray[playerPosition].vertices.First.Value)
        {
            firstPolygonVertices.AddLast(gameActivity.gamePlayArea.verticesNo
de.Value);

            if(gameActivity.gamePlayArea.verticesNode.Next != null)
            {
                gameActivity.gamePlayArea.verticesNode =
gameActivity.gamePlayArea.verticesNode.Next;
            }
            else
            {
                gameActivity.gamePlayArea.verticesNode =
gameActivity.gamePlayArea.vertices.First;
            }
        }
        else
        {
            break;
        }
    }
}

```

```

private void AddSecondPolygon(int playerPosition, LinkedList<LatLng>
secondPolygonVertices)
{
    if(gameActivity.gamePlayArea.vertices.Find(gameActivity.playerArray[playerPosit
ion].vertices.Last.Value).Previous != null)
    {
        gameActivity.gamePlayArea.verticesNode =
gameActivity.gamePlayArea.vertices.Find(gameActivity.playerArray[playerPosition
].vertices.Last.Value).Previous;
    }
    else
    {
        gameActivity.gamePlayArea.verticesNode =
gameActivity.gamePlayArea.vertices.Last;
    }

    while(true)
    {
        if(gameActivity.gamePlayArea.verticesNode.Value !=
gameActivity.playerArray[playerPosition].vertices.First.Value)
        {
            secondPolygonVertices.AddLast(gameActivity.gamePlayArea.verticesN
ode.Value);

            if(gameActivity.gamePlayArea.verticesNode.Previous != null)
            {
                gameActivity.gamePlayArea.verticesNode =
gameActivity.gamePlayArea.verticesNode.Previous;
            }
            else
            {
                gameActivity.gamePlayArea.verticesNode =
gameActivity.gamePlayArea.vertices.Last;
            }
        }
        else
        {
            break;
        }
    }
}

```

```

private void TestAreas(int playerPosition)
{
    double firstArea = Static.Maths.PolygonArea(new
LinkedList<LatLng>(gameActivity.gamePlayArea.polygons.Last.Previous.Value.Points));

    double secondArea = Static.Maths.PolygonArea(new
LinkedList<LatLng>(gameActivity.gamePlayArea.polygons.Last.Value.Points));

    if(firstArea <= secondArea)
    {
        AddFirstArea(playerPosition, firstArea, secondArea);
    }
    else
    {
        AddSecondArea(playerPosition, secondArea, firstArea);
    }

    UpdatePolygons(playerPosition);
}

private void AddFirstArea(int playerPosition, double takenArea, double leftArea)
{
    gameActivity.gamePlayArea.polygons.First.Value.Points =
gameActivity.gamePlayArea.polygons.Last.Value.Points;

    gameActivity.gamePlayArea.polygons.Remove(gameActivity.gamePlayArea.polygons.La
st.Value);

    UpdateScore(playerPosition, takenArea, leftArea);
}

private void AddSecondArea(int playerPosition, double takenArea, double leftArea)
{
    gameActivity.gamePlayArea.polygons.First.Value.Points =
gameActivity.gamePlayArea.polygons.Last.Previous.Value.Points;

    gameActivity.gamePlayArea.polygons.Remove(gameActivity.gamePlayArea.polygons.La
st.Previous.Value);

    UpdateScore(playerPosition, takenArea, leftArea);
}

private void UpdateScore(int playerPosition, double takenArea, double leftArea)
{
    gameActivity.area = (int)((leftArea / gameActivity.initialArea) * 100);

    gameActivity.areaTextView.Text = "Area: " + gameActivity.area.ToString();

    gameActivity.playerArray[playerPosition].score += (int)((takenArea /
gameActivity.initialArea) * 100);

    if(playerPosition == gameActivity.playerPosition)
    {
        gameActivity.scoreTextView.Text = "Score: " +
gameActivity.playerArray[gameActivity.playerPosition].score.ToString();
    }
}

```

```
private void UpdatePolygons(int playerPosition)
{
    gameActivity.gamePlayArea.vertices = new
LinkedList<LatLng>(gameActivity.gamePlayArea.polygons.First.Value.Points);

    gameActivity.gamePlayArea.vertices.RemoveLast();

    gameActivity.gamePlayArea.polygons.Last.Value.FillColor = Color.HSVToColor(new
float[] { utilities.Colour(playerPosition), 1.0f, 1.0f });
}
```


Appendix G: Polygon Tessellation Code

```
//Triangulate the polygon.

private static List<Game.Triangle> Triangulate(LinkedList<LatLng> vertices)
{
    //Copy the points into a scratch array.

    int num_points = vertices.Count;

    LatLng[] pts = new LatLng[num_points];

    vertices.CopyTo(pts, 0);

    //Make room for the triangles.

    List<Game.Triangle> triangles = new List<Game.Triangle>();

    //While the copy of the polygon has more than

    //three points, remove an ear.

    while(pts.Length > 3)
    {
        //Remove an ear from the polygon.

        RemoveEar(ref pts, triangles);
    }

    //Copy the last three points into their own triangle.

    triangles.Add(AddTriangle(pts[0], pts[1], pts[2]));

    return triangles;
}

private static Game.Triangle AddTriangle(LatLng a, LatLng b, LatLng c)
{
    Game.Triangle triangle = new Game.Triangle();

    triangle.vertices = new LatLng[3];

    triangle.vertices[0] = a;

    triangle.vertices[1] = b;

    triangle.vertices[2] = c;

    return triangle;
}
```

```

//Remove an ear from the polygon and
//add it to the triangles array.
private static void RemoveEar(ref LatLng[] pts, List<Game.Triangle> triangles)
{
    //Find an ear.

    int a = 0;

    int b = 0;

    int c = 0;

    FindEar(pts, ref a, ref b, ref c);

    //Create a new triangle for the ear.

    triangles.Add(AddTriangle(pts[a], pts[b], pts[c]));

    //Remove the ear from the polygon.

    RemovePointFromArray(ref pts, b);
}

//Find the indexes of three points that form an "ear."
private static void FindEar(LatLng[] pts, ref int a, ref int b, ref int c)
{
    int num_points = pts.Length;

    for(a = 0; a < num_points; a++)
    {
        b = (a + 1) % num_points;

        c = (b + 1) % num_points;

        if(FormsEar(pts, a, b, c))
        {
            return;
        }
    }

    a--;

    //We should never get here because there should

    //always be at least two ears.

    Debug.Assert(false);
}

```

```

//Return true if the three points form an ear.

private static bool FormsEar(LatLng[] pts, int a, int b, int c)
{
    //See if the angle ABC is concave.

    if(GetAngle(pts[a].Latitude, pts[a].Longitude, pts[b].Latitude,
pts[b].Longitude, pts[c].Latitude, pts[c].Longitude) > 0)
    {
        //This is a concave corner so the triangle

        //cannot be an ear.

        return false;
    }

    //Make the triangle A, B, C.

    Game.Triangle triangle = AddTriangle(pts[a], pts[b], pts[c]);

    //Check the other points to see

    //if they lie in triangle A, B, C.

    for(int i = 0; i < pts.Length; i++)
    {
        if((i != a) && (i != b) && (i != c))
        {
            if(PointInTriangle(triangle, pts[i]))
            {
                //This point is in the triangle

                //do this is not an ear.

                return false;
            }
        }
    }

    //This is an ear.

    return true;
}

```

```

//Return true if the point is in the polygon.

private static bool PointInTriangle(Game.Triangle triangle, LatLng point)
{
    //Get the angle between the point and the
    //first and last vertices.

    int max_point = triangle.vertices.Length - 1;

    double total_angle = GetAngle(triangle.vertices[max_point].Latitude,
triangle.vertices[max_point].Longitude, point.Latitude, point.Longitude,
triangle.vertices[0].Latitude, triangle.vertices[0].Longitude);

    //Add the angles from the point
    //to each other pair of vertices.

    for(int i = 0; i < max_point; i++)
    {
        total_angle += GetAngle(triangle.vertices[i].Latitude,
triangle.vertices[i].Longitude, point.Latitude, point.Longitude,
triangle.vertices[i + 1].Latitude, triangle.vertices[i + 1].Longitude);
    }

    //The total angle should be 2 * PI or -2 * PI if
    //the point is in the polygon and close to zero
    //if the point is outside the polygon.

    return (Math.Abs(total_angle) > 0.000001);
}

//Remove point target from the array.

private static void RemovePointFromArray(ref LatLng[] pts, int target)
{
    LatLng[] newPTS = new LatLng[pts.Length - 1];

    Array.Copy(pts, 0, newPTS, 0, target);

    Array.Copy(pts, target + 1, newPTS, target, pts.Length - target - 1);

    pts = newPTS;
}

```

Appendix H: Wandering Code

```
public void RunHazards(LinkedList<LatLng> polygonVertices)
{
    if(gameActivity.gameType == "Single Player" || gameActivity.gameType == "Host")
    {
        gameActivity.RunOnUiThread(() => CheckPosition(polygonVertices));
    }
    else
    {
        if(gameActivity.gameType == "Join")
        {
            gameActivity.RunOnUiThread(() =>
            CheckPlayerInterception(circle.Radius));
        }
    }
}
```

```

private void CheckPosition(LinkedList<LatLng> polygonVertices)
{
    LatLng position = circle.Center;

    double radius = circle.Radius;

    if(!Static.Maths.PointInPolygon(polygonVertices, position))
    {
        gameActivity.circle.Center =
        utilities.FindCirclePosition(polygonVertices, radius);

        SetCircle(gameActivity.circle);

        SetDegrees();
    }
    else
    {
        LatLng[] interceptionVertex;

        UpdatePosition(radius);

        if(utilities.CircleIntersectPolygon(polygonVertices, circle.Center,
        radius, out interceptionVertex))
        {
            gameActivity.circle.Center = position;

            SetCircle(gameActivity.circle);

            ReverseUnitVector(interceptionVertex);

            if(attempts >= 10)
            {
                gameActivity.circle.Center =
                utilities.FindCirclePosition(polygonVertices, radius);

                SetCircle(gameActivity.circle);

                SetDegrees();
            }
            else
            {
                attempts++;
            }
        }
        else
        {
            attempts = 0;
        }
    }
}

```

```

private void UpdatePosition(double radius)
{
    LatLng degreesUnitVector = Static.Maths.DegreesToUnitVector(degrees);

    circle.Center = new LatLng(circle.Center.Latitude + (degreesUnitVector.Latitude
* velocity), circle.Center.Longitude + (degreesUnitVector.Longitude * velocity));

    gameActivity.circle.Center = circle.Center;

    CheckPlayerInterception(radius);

    UpdateUnitVector();
}

private void CheckPlayerInterception(double radius)
{
    LinkedList<LatLng> pathVertices = new LinkedList<LatLng>(); ;

    bool deathBool;

    LatLng[] interceptionVertex;

    for(int i = 0; i < gameActivity.numberOfPlayers; i++)
    {
        gameActivity.CopyVertices(pathVertices,
gameActivity.playerArray[i].vertices);

        gameActivity.CopyBool(out deathBool,
gameActivity.playerArray[i].deathBool);

        if(pathVertices.First != null && pathVertices.First.Next != null)
        {
            if(utilities.CircleIntersectPolygon(pathVertices, circle.Center,
radius, out interceptionVertex) && !deathBool)
            {
                gameActivity.KillPlayer(i);
            }
        }
    }
}

```

```

private void UpdateUnitVector()
{
    Random random = new Random();

    int acceleration = random.Next(0, 91) - 45;

    int temporaryDegrees = degrees + acceleration;

    if(temporaryDegrees > 360)
    {
        degrees = acceleration - (360 - temporaryDegrees);
    }
    else
    {
        if(temporaryDegrees < 0)
        {
            degrees = 360 - (acceleration - temporaryDegrees);
        }
        else
        {
            degrees = temporaryDegrees;
        }
    }
}

```



```

private void ReverseUnitVector(LatLng[] interceptionVertex)
{
    LatLng interceptionNormalVector = new LatLng(interceptionVertex[1].Longitude -
interceptionVertex[0].Longitude, -(interceptionVertex[1].Latitude -
interceptionVertex[0].Latitude));

    double interceptionNormalVectorMagnitude =
Math.Sqrt(Math.Pow(interceptionNormalVector.Latitude, 2) +
Math.Pow(interceptionNormalVector.Longitude, 2));

    interceptionNormalVector = new
LatLng(Math.Abs(interceptionNormalVector.Latitude / interceptionNormalVectorMagnitude)
* -1, Math.Abs(interceptionNormalVector.Longitude / interceptionNormalVectorMagnitude)
* -1);

    if(interceptionNormalVector.Latitude == 0)
    {
        interceptionNormalVector.Latitude = 1;
    }

    if(interceptionNormalVector.Longitude == 0)
    {
        interceptionNormalVector.Longitude = 1;
    }

    LatLng degreesUnitVector = Static.Maths.DegreesToUnitVector(degrees);

    degreesUnitVector = new LatLng(degreesUnitVector.Latitude *
interceptionNormalVector.Latitude, degreesUnitVector.Longitude *
interceptionNormalVector.Longitude);

    double degreesUnitVectorMagnitude =
Math.Sqrt(Math.Pow(degreesUnitVector.Latitude, 2) +
Math.Pow(degreesUnitVector.Longitude, 2));

    degreesUnitVector = new LatLng(degreesUnitVector.Latitude /
degreesUnitVectorMagnitude, degreesUnitVector.Longitude / degreesUnitVectorMagnitude);

    degrees = Static.Maths.UnitVectorToDegrees(degreesUnitVector);
}

```

Appendix I: Initial Task List

#	Task Name	Description	Duration (weeks)
1	Research	Conduct research that will aid in the writing of reports and initial designing of the project	10
2	Initial report	Write the initial report deliverable	2
3	Create server client library	Create a client library that is capable of interfacing with the server	5
4	Create TCP server	Create a TCP server that can accept TCP packets from the client	2
5	Add UDP to server	Add UDP to the server that can be used to identify the IP of the server	1
6	Add multithreading capabilities to server	Add multithreading to the server so that it is capable of accepting more than one client request at a time	2
7	Create main menu for game	Create a main menu for the game that will be displayed when the game is started and between every game instance. The main menu should display all options for game types and settings	1
8	Add game screen and assets to game	Add a game screen to the game that is displayed once the play game option is selected and also add assets to the game to be used to display player characters	2
9	Add map to game screen	Add a suitable map to the game screen	3
10	Add translations and scaling to map	Add translations and scaling to the map so that it is possible to move the map around and zoom in and out	2
11	Add player character and	Make the player character move as the player moves, this should work via GPS	2

	movements to map		
12	Interim report	Write the interim report deliverable	3
13	Add client calls to store and recall player positions from server	Make the game send its current location to the server at a reasonable interval and also make it so that the game requests the location of every other player	3
14	Final report	Write the final report deliverable	14
15	Add tracking data and bounds of playing field to game	Make it so that the game then draws all the players at the correct locations and that it is possible to create the area of play	2
16	Add taking mechanics from tracking data to game	Make it so that when a player completes a run from one side of the playing area to another they take the smallest area for their own team	2
17	Add killing mechanics from tracking data	Make it so that if a player crosses the track of another active player one of the players dies	2
18	Add scoring data to game	Make it so that the score of all players is tracked based on the area of land taken	2
19	Add team mode to game	Make it so that players can play in teams	3
20	Add multigame server	Make it so that multiple game instances can run on one server	3

Appendix J: Initial Time Plan

#	Task Name	University Calendar Weeks																																			
		4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36			
1	Research																																				
2	Initial report				D																																
3	Create server client																																				
4	Create TCP server																																				
5	Add UDP to server																																				
6	Add multithreading capabilities to server																																				
7	Create main menu for game																																				
8	Add game screen and assets to game																																				
9	Add map to game screen																																				
10	Add translations and scaling to map																																				
11	Add player character and movements to map																																				
12	Interim report																																				

Appendix K: Interim Task List

#	Task Name	Description	Duration (weeks)
1	Research	Conduct research that will aid in the writing of reports and initial designing of the project	9
2	Initial report	Write the initial report deliverable	2
3	Create server client library	Create a client library that is capable of interfacing with the server	5
4	Create TCP server	Create a TCP server that can accept TCP packets from the client	2
5	Create UDP server	Create a UDP server that can accept UDP packets from the client and can be used to identify the IP of the server	1
6	Add multithreading capabilities to server	Add multithreading to the server so that it is capable of accepting more than one client request at a time	2
7	Add logging capabilities to the server	Add logging to the server so that it is capable of recovering from a fatal error	2
8	Work on other projects	Time set aside to work on other projects	6
9	Interim report	Write the interim report deliverable	3
10	Create main menu for game	Create a main menu for the game that will be displayed when the game is started and between every game instance. The main menu should display all options for game types and settings	1
11	Add game screen and assets to game	Add a game screen to the game that is displayed once the play game option is selected and also add assets to the game to be used to display player characters	2
12	Add map to game screen	Add a suitable map to the game screen	2

13	Add translations and scaling to map	Add translations and scaling to the map so that it is possible to move the map around and zoom in and out	2
14	Add player character and movements to map	Make the player character move as the player moves, this should work via GPS	2
15	Add non-player entities	Add a non-player entity that is capable of killing the player, this entity should move following a randomly generated meandering path	2
16	Final report	Write the final report deliverable	14
17	Add client calls to store and recall player positions from server	Make the game send its current location to the server at a reasonable interval and also make it so that the game requests the location of every other player	2
18	Add tracking data and bounds of playing field to game	Make it so that the game then draws all the players at the correct locations and that it is possible to create the area of play	2
19	Add taking mechanics from tracking data to game	Make it so that when a player completes a run from one side of the playing area to another they take the smallest area for their own team	2
20	Add killing mechanics from tracking data	Make it so that if a player crosses the path of another active player one of the players dies	2
21	Add scoring data to game	Make it so that the score of all players is tracked based on the area of land taken	2

22	Work on other projects	Time set aside to work on other projects	7
23	Add team mode to game	Make it so that players can play in teams	3

Appendix L: Interim Time Plan

#	Task Name	University Calendar Weeks																																			
		4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36			
1	Research																																				
2	Initial report				D																																
3	Create server client																																				
4	Create TCP server																																				
5	Add UDP to server																																				
6	Add multithreading capabilities to server																																				
7	Add logging capabilities to the server																																				
8	Work on other projects																																				
9	Interim report																		D																		
10	Create main menu for game																																				
11	Add game screen and assets to game																																				
12	Add map to game screen																																				
13	Add translations																																				

Appendix M: Risk Analysis

Risk	Current Risk			How to Avoid	How to Recover	Residual Risk		
	Severity (L/M/H)	Likelihood (L/M/H)	Significance (Severity, Likelihood)			Severity (L/M/H)	Likelihood (L/M/H)	Significance (Severity, Likelihood)
Data loss	H	M	HM	Keep backups	Reinstate from backups	L	M	LM
Loss of backups	H	L	HL	Keep multiple backups	Use alternate backup	L	L	LL
Underestimate workload	H	M	HM	Regularly review progress against Time Plan	Invest more time into work, possible reduction of objectives	H	L	HL
Critical error in deliverable	H	M	HM	Perform adequate research	Debug code	H	L	HL
Skill Risk	M	M	MM	Perform adequate training	Invest more time into research	L	L	LL
Scope Creep	M	H	MH	Fully define scope	Define scope at current point	M	L	ML
Inefficient Program Performance	H	L	HL	Spend time testing code	Remove extraneous features	M	L	ML

Server Crashes	M	M	MM	Attempt to optimize server code	Implement alternative servers	L	L	LL
Incompatible with target device	H	L	HL	Create program with target device specifications in mind	Create alternative version for target device	L	L	LL
Medical emergency	H	L	HL	Care for developers health	Comment code regularly so that it is well understood	M	L	ML
Development software unavailable during demonstration	H	H	HH	Place key files into a backup	Download key files	L	L	LL
Development software unavailable during development	H	L	HL	Ensure development software is installed on development computer	Keep a backup computer to use in the case that the development software cannot be installed on the main computers	M	L	ML

Backup computer insufficient to develop on	H	M	HM	Ensure development software is installed on development computer so the backup computer is unnecessary	Upgrade the backup computer	H	L	HL
Insufficient funds to upgrade backup computer	H	H	HH	A way to avoid this risk would be beyond the scope of this project.	Change the development software being used	H	H	HH
User injured while using application	H	M	HM	Only allow authorized access to application while in development and provide warning messages while in play	Medical attention may be required if a user is injured while using the application	H	L	HL

Acknowledgements

References

- Ackermann, F., 1994. PRACTICAL EXPERIENCE WITH GPS SUPPORTED AERIAL TRIANGULATION. *The Photogrammetric Record*, 14(84), pp. 860-874.
- Ali, D. F. A., Mustafa, F. a. & Fdolesid, M. O. M., n.d. *Automatic Handover Control for Distributed Load Balancing in Mobile Communication*, s.l.: Future University, Faculty of Engineering, Department of Computer Engineering.
- Anna, 2016. *HOW TO BUILD A MOBILE APP WITH GEOLOCATION?*. [Online]
Available at: <https://theappsolutions.com/blog/development/develop-app-with-geolocation/>
[Accessed 20 April 2017].
- Apple, 2016. *The powerful programming language that is also easy to learn..* [Online]
Available at: <https://developer.apple.com/swift/>
[Accessed 12 October 2016].
- Apple, 2017. *Tools you'll love to use..* [Online]
Available at: <https://developer.apple.com/xcode/ide/>
[Accessed 11 January 2017].
- Baccelli, F., Mathieu, F., Norros, I. & Varloot, R., 2013. *Can P2P Networks be Super-Scalable*. [Online]
Available at: <https://arxiv.org/pdf/1304.6489.pdf>
[Accessed 14 November 2016].
- Balagtas-Fernandez, F., Forrai, J. & Hussmann, H., 2009. *Evaluation of User Interface Design and Input Methods for Applications on Mobile Touch Screen Devices*. Uppsala, Springer, Berlin, Heidelberg.
- Berard, E. V., 2015. *Abstraction, Encapsulation, and Information Hiding*. [Online]
Available at: <http://www.tonymarston.co.uk/php-mysql/abstraction.txt>
[Accessed 17 January 2017].
- Bernstein, P. A. & Goodman, N., 1981. Concurrency Control in Distributed Database Systems. *ACM Computing Surveys*, 13(2), pp. 185-221.
- Brown, A. K. & Sturza, M. A., 1995. *GPS tracking system*. United States of America, Patent No. US 07/800,850.
- Crato, N., 2010. How GPS Works. In: *Figuring it Out*. Lisboa: Springer-Verlag Berlin Heidelberg, pp. 49-52.
- Credits, Extra, 2016. *Improving on Pokemon GO - Making Better Augmented Reality Games - Extra Credits*. [Online]
Available at: <https://www.youtube.com/watch?v=94KwB205DDk>
[Accessed 11 October 2016].
- Dempsey, C., 2013. *War2Map – A Role-Playing Google Maps Game*. [Online]
Available at: <https://www.gislounge.com/war2map-role-playing-google-maps-game/>
[Accessed 16 January 2017].
- Diffen, n.d. *TCP vs. UDP*. [Online]
Available at: http://www.diffen.com/difference/TCP_vs_UDP
[Accessed 11 October 2016].

Farcic, V., 2014. *Software Development Models: Iterative and Incremental Development*. [Online]

Available at: <https://technologyconversations.com/2014/01/21/software-development-models-iterative-and-incremental-development/>

[Accessed 2 May 2017].

Funk, B., 2015. *Microsoft's Arrow launcher personalizes the Android experience*. [Online]

Available at: <http://techreport.com/news/29254/microsoft-arrow-launcher-personalizes-the-android-experience>

[Accessed 18 January 2017].

Garmin, 2008. *Garmin Proprietary*. [Online]

Available at: http://www.garmin.com/support/pdf/NMEA_0183.pdf

[Accessed 20 April 2017].

Google, 2016. *FusedLocationProviderApi*. [Online]

Available at:

<https://developers.google.com/android/reference/com/google/android/gms/location/FusedLocationProviderApi>

[Accessed 19 January 2017].

Google, 2017. *Ingress*. [Online]

Available at: <https://play.google.com/store/apps/details?id=com.nianticproject.ingress>

[Accessed 25 April 2017].

Google, 2017. *The Google Maps Geolocation API*. [Online]

Available at: <https://developers.google.com/maps/documentation/geolocation/intro>

[Accessed 19 January 2017].

Google, n.d. *Getting Started with the NDK*. [Online]

Available at: <https://developer.android.com/ndk/guides/index.html>

[Accessed 11 October 2016].

Howcroft, D. & Carroll, J., 2000. *A Proposed Methodology for Web Development*. Salford, European Conference on Information Systems.

Ian A.R. Hulbert, J. F., 2001. The accuracy of GPS for wildlife telemetry and habitat mapping. *Journal of Applied Ecology*, 38(4), pp. 869-878.

IBM, n.d. *How the Watchdog application works*. [Online]

Available at:

https://www.ibm.com/support/knowledgecenter/en/SSPK3V_6.3.0/com.ibm.swg.im.soliddb.ha.doc/doc/s0000193.how.the.watchdog.application.works.html

[Accessed 30 April 2017].

IT Knowledge portal, 2017. *Software Development Methodologies*. [Online]

Available at: <http://www.itinfo.am/eng/software-development-methodologies/>

[Accessed 2 May 2017].

Jeremy, 2014. *Using Build Events in Visual Studio to Make Life Easier*. [Online]

Available at: <https://jeremybytes.blogspot.co.uk/2014/02/using-build-events-in-visual-studio-to.html>

[Accessed 29 April 2017].

Kaplan, E. D. & Hegarty, C. J., 2006. *Understanding GPS Principles and Applications*. 2nd ed. Norwood: Artech House.

- Larman, C. & Basili, V., 2003. Iterative and incremental developments. a brief history. *Computer*, 36(6), pp. 47-56.
- Lawson, V., 2016. *A forensic examination of Pokemon Go*, Ann Arbor: ProQuest Dissertations Publishing.
- Madhavan, A., 2016. *Pokémon GO: A Lesson In Product Rebranding*. [Online] Available at: <https://amplitude.com/blog/2016/07/12/pokemon-go-lesson-product-rebranding/> [Accessed 20 April 2017].
- Mey, M. D., n.d. *Network discovery using UDP Broadcast (Java)*. [Online] Available at: <http://michioldemey.be/blog/network-discovery-using-udp-broadcast/> [Accessed 11 October 2016].
- Montemagno, J., 2016. *.NET Standard Library Support for Xamarin*. [Online] Available at: <https://blog.xamarin.com/net-standard-library-support-for-xamarin/> [Accessed 11 October 2016].
- Mooney, F. W., 1985. Terrestrial Evaluation of the GPS Standard Positioning Service. *Navigation*, 32(4), pp. 351-369.
- Niantic, 2016. *Ingress*. [Online] Available at: <https://www.ingress.com/> [Accessed 11 October 2016].
- Niantic, 2016. *Pokémon GO*. [Online] Available at: <http://www.pokemongo.com/en-uk/> [Accessed 11 October 2016].
- Noble, J., 2002. Visualising Objects: Abstraction, Encapsulation, Aliasing, and Ownership. In: S. Diehl, ed. *Software Visualization*. Wellington: Springer Berlin Heidelberg, pp. 58-72.
- O'Sullivan, C., 2015. *A Tale of Two Platforms: Designing for Both Android and iOS*. [Online] Available at: <https://webdesign.tutsplus.com/articles/a-tale-of-two-platforms-designing-for-both-android-and-ios--cms-23616> [Accessed 20 April 2017].
- openclipart, 2014. *GPS satellites - trilateration*. [Online] Available at: <https://openclipart.org/detail/191659/gps-satellites-trilateration> [Accessed 19 January 2017].
- Ordnance Survey, 2017. *GETTING OUTSIDE WITH POKEMON GO*. [Online] Available at: <https://www.ordnancesurvey.co.uk/getoutside/guides/getting-outside-with-pokemon-go/> [Accessed 20 April 2017].
- Park Productions, 2015. *P.A.U.L.A - Park Arcade Unit List Array*. [Online] Available at: <http://www.parkproductions.co.uk/area/games/games.htm> [Accessed 18 January 2017].
- Philips, B., 2014. *The P2P Witch Hunt*. [Online] Available at: <http://blog.peer5.com/the-p2p-witch-hunt/> [Accessed 11 October 2016].
- Pilato, M., Collins-Sussman, B. & Fitzpatrick, B., 2002. *Version Control with Subversion*. 2nd ed. Sebastopol: O'Reilly Media Inc.

- Posey, B., 2000. *Understanding the differences between client/server and peer-to-peer networks*. [Online]
Available at: <http://www.techrepublic.com/article/understanding-the-differences-between-client-server-and-peer-to-peer-networks/>
[Accessed 11 October 2016].
- R, A., 2013. *Managing Computers with Automation*. [Online]
Available at: http://techthoughts.typepad.com/managing_computers/heartbeat/
[Accessed 30 April 2017].
- Rouse, M., 2006. *native code*. [Online]
Available at: <http://searchmicroservices.techtarget.com/definition/native-code>
[Accessed 25 April 2017].
- Schach, S. R., 2007. *Object-Oriented and Classical Software Engineering*. 7th ed. s.l.:The McGraw-Hill Companies.
- Scholtz, J., 1992. The role of planning in learning a new programming language. *International Journal of Man-Machine Studies*, 37(2), pp. 191-214.
- Schwaber, K., 1997. *SCRUM Development Process*. London, s.n.
- Seguin, K., 2014. *How unreliable is UDP?*. [Online]
Available at: <http://openmymind.net/How-Unreliable-Is-UDP/>
[Accessed 10 January 2017].
- Shellnut, B., Knowlton, A. & Savage, T., 1999. Applying the ARCS model to the design and development of computer-based modules for manufacturing engineering courses. *Educational Technology Research and Development*, 47(2), pp. 100-110.
- Sinicki, A., 2016. *Developing for Android vs developing for iOS – in 5 rounds*. [Online]
Available at: <http://www.androidauthority.com/developing-for-android-vs-ios-697304/>
[Accessed 11 October 2016].
- Snader, J. C., n.d. *Effective TCP/IP Programming: 44 Tips to Improve Your Network Programs*. s.l.:s.n.
- Sparrow, P., 2017. *Client Server Network : Advantages and Disadvantages*. [Online]
Available at: <http://www.ianswer4u.com/2011/05/client-server-network-advantages-and.html#axzz4fFqNt6qT>
[Accessed 25 April 2017].
- Spikejumper2, 2017. *Are lives in videogames obsolete?*. [Online]
Available at: <http://gdforum.freeforums.net/thread/46514/lives-videogames-obsolete>
[Accessed 19 April 2017].
- Statista, 2016. *Global mobile OS market share in sales to end users from 1st quarter 2009 to 1st quarter 2016*. [Online]
Available at: <https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/>
[Accessed 11 October 2016].
- Stenovec, T., 2015. *Apple's iOS is beating Android where it matters*. [Online]
Available at: <http://uk.businessinsider.com/ios-users-spend-more-than-android-users-2015-12?r=US&IR=T>
[Accessed 20 April 2017].

Sunny, K., 2016. *Distribution vs Profitability – the future of hotel revenue management*. [Online]

Available at: <http://www.hospitalitynet.org/news/4077889.html>
[Accessed 20 April 2017].

System 16, 2014. *TAITO QIX HARDWARE*. [Online]

Available at: <http://www.system16.com/hardware.php?id=633>
[Accessed 11 October 2016].

Taing, N., 2011. *TCP UDP and RMI Performance Evaluation*. [Online]

Available at: <http://lycog.com/performance-evaluation/tcp-udp-rmi-performance-evaluation/>
[Accessed 25 April 2017].

Tarasenko, N., 2009. *Geostationary Satellites and the GPS Constellation*. [Online]

Available at: http://ccar.colorado.edu/asen5050/projects/projects_2009/tarasenko/
[Accessed 19 January 2017].

TechTarget, 2014. *TCP (Transmission Control Protocol)*. [Online]

Available at: <http://searchnetworking.techtarget.com/definition/TCP>
[Accessed 11 October 2016].

TechTarget, 2015. *UDP (User Datagram Protocol)*. [Online]

Available at: <http://searchsoa.techtarget.com/definition/UDP>
[Accessed 11 October 2016].

The International Arcade Museum, Museum of the Game, 2016. *Qix*. [Online]

Available at: http://www.arcade-museum.com/game_detail.php?game_id=9185
[Accessed 12 October 2016].

Vogel, L., 2016. *Android development with Android Studio - Tutorial*. [Online]

Available at: <http://www.vogella.com/tutorials/Android/article.html>
[Accessed 18 January 2017].

Whelan, F., 2014. *Review: Google's Ingress makes every day a glorious battle*. [Online]

Available at: <http://www.independent.ie/entertainment/games/mobile/review-googles-ingress-makes-every-day-a-glorious-battle-30459932.html>
[Accessed 20 April 2017].

Wikipedia, 2016. *Class Diagram*. [Online]

Available at: https://en.wikipedia.org/wiki/Class_diagram
[Accessed 16 January 2017].

Wikipedia, 2017. *Android Studio*. [Online]

Available at: https://en.wikipedia.org/wiki/Android_Studio
[Accessed 11 January 2017].

Wikipedia, 2017. *Ingress (Video game)*. [Online]

Available at: [https://en.wikipedia.org/wiki/Ingress_\(video_game\)](https://en.wikipedia.org/wiki/Ingress_(video_game))
[Accessed 11 January 2017].

Wikipedia, 2017. *Iterative and incremental development*. [Online]

Available at: https://en.wikipedia.org/wiki/Iterative_and_incremental_development
[Accessed 2 May 2017].

Wikipedia, 2017. *Microsoft Visual Studio*. [Online]
Available at: https://en.wikipedia.org/wiki/Microsoft_Visual_Studio
[Accessed 11 January 2017].

Wikipedia, 2017. *Pokemon GO*. [Online]
Available at: https://en.wikipedia.org/wiki/Pok%C3%A9mon_Go
[Accessed 11 January 2017].

Wikipedia, 2017. *Software development process*. [Online]
Available at: https://en.wikipedia.org/wiki/Software_development_process
[Accessed 2 May 2017].

Xamarin Inc., 2016. *Building Cross Platform Applications*. [Online]
Available at: https://developer.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/
[Accessed 11 October 2016].

Xamarin Inc., 2016. *Using Native Libraries*. [Online]
Available at:
https://developer.xamarin.com/guides/android/advanced_topics/using_native_libraries/
[Accessed 11 October 2016].

Xamarin, 2017. *Introduction to Location Services and the Fused Location Provider*. [Online]
Available at:
https://developer.xamarin.com/guides/android/platform_features/maps_and_location/location/
[Accessed 19 January 2017].

Zhang, Y. & Soong, B.-H., 2006. Performance of Mobile Networks with. *IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS*, 5(5), pp. 990-995.

Zhao, Y., 2002. Standardization of mobile phone positioning for 3G systems. *IEEE Communications Magazine*, 7 August, pp. 108-116.