Mixed Reality Tabletop War Game Assistant

Submitted April 2024, in partial fulfilment of

the conditions for the award of the degree:
BSc Hons Computer Science

20363169
School of Computer Science
University of Nottingham

I hereby declare that this dissertation is all my own work, except as indicated in the text:

Signature: NB
Date: 13/04/24

I hereby declare that I have all necessary rights and consents to publicly distribute this dissertation via the University of Nottingham's e-dissertation archive.*

Table of Contents

In this document, there are 9153 words all up.

1. Abstract

2. Introduction and Motivation

Tabletop war-gaming is a popular hobby that is quickly gaining popularity, however, with a high barrier to entry, it remains niche and inaccessible to many. The rules to tabletop war-games can be complex and difficult to learn. This can be daunting for new players, putting them off the hobby as well as causing disagreement between seasoned players over rules interpretations.

Some of the most popular war-gaming systems are produced by *Games Workshop* [1]. One of their more popular systems, *Warhammer 40k*, has a core rule-book of 60 pages [2] and the simplified version of another game system, *Kill Team*, is a rather dense three page spread [3]. This complexity can be off putting to new players. Many tabletop / boardgames suffer from a players first few games taking significantly longer than is advertised due to needing to constantly check the rules.

As well as this, new players are likely to take longer to make decisions as they are not familiar with the game's mechanics of what constitutes a "good" move (sometimes referred to as "analysis paralysis"). This can be exacerbated by the game's reliance on dice rolls to determine the outcome of actions. Meaning that seemingly "optimal" moves do not always result in favourable outcomes, causing an extended learning period for an already complex game.

Kill Team is a miniature war-game known as a "skirmish" game. This means the game is played on a smaller scale with only ~20 miniatures on the table at one time. The aim of the game is to compete over objectives on the board for points. Each player takes turns activating a miniature and performing actions with it. These a selection of actions are: moving to another location, shooting at an enemy, melee combat and capturing an objective. The game uses dice to determine the results of your models engaging in combat with each other with the required rolls being determined by the statistics of each "operative" (a miniature) involved and the terrain.

Figure 1: An example of the *Kill Team* tabletop game using the *Gallowdark* terrain. The terrain we will focus on here are the flat walls and pillars. [4]

Video games help on-board new players by having the rules enforced by the game itself. This project aims to bring a similar methodology to tabletop war-gaming, specifically the *Kill Team Lite* [3] system

using the *Gallowdark* setting. The *Kill Team Lite* rules are publicly available from *Games Workshop's* website and is designed to be played on a smaller scale to other war games, making it a good candidate for a proof of concept. As well as this, the *Gallowdark* [5] setting streamlines the terrain used and removes verticality from the game, making implementation much simpler.

Developing a system that can digitally represent a physical *Kill Team* board would allow for the creation of tools to assist players. For example, a digital game helper would remove the burden of rules enforcement from the players and onto the system, allowing players to focus on the game itself or allow players to make more informed game decisions by being able to preview the options available to them. This could also be utilised to record a game and view a replay or be used for content creation to make accurate board representations to viewers with digital effects.

2.1. Project Intention and Background

This project will focus on the development of a system that can track the position of miniature models and terrain pieces on a *Kill Team* board which can subsequently be displayed digitally. From here, we aim to implement proof of concept visualisation of a few select game rules on the digital board to demonstrate that the tracking system provides the necessary information to process said rules.

To determine the specific goals for this project, it is important to provide some more context on the gameplay and community surrounding *Kill Team*. As this is a very complex game, we will be omitting a large percentage of the rules, focusing on the topics that will have an impact on the overall design or provide unique challenges.

2.1.1. Gameboard Components

Before looking at the game rules, we will first break down the physical components of the gameboard to determine what needs to be tracked and problems that may arrise from this.

A game of *Kill Team* is comprised of two players, each with a group of "operatives" (miniatures) referred to as a "Kill Team". Each Kill Team has it's own unique rules, with each operative having it's own set of unique statistics and special abilities. A miniature is comprised of a circular base with a model placed on top. The diameter of the base is either 25mm, 28.5mm, 32mm, 40mm or occasionally 50mm.

Figure 2: An example of a *Kasrkin Kill Team* operative on a 28.5mm base [6]. The height of the operative is ~35mm excluding the base. This project will focus on getting the system functional with the *Kasrkin* models on 28.5mm bases.

It is worth noting that is is common for models to extend past the edge of their base as seen in Figure 2. Whilst the example above is somewhat minor in its overlap, different models can be more extreme in their extension. As a result of this the system will need to be able to locate and identify an operative from only a partial view of the base, or its surroundings, from a bird's eye view. The detection system will either need to be above the board if using visual detection methods or below the board if using, for example, an RFID method. This is due to the terrain potentialy surrounding an operative, so having a camera on each side of the board will not guarantee it is visible.

The game is played on a 30″ by 22″ board, referred to as a "Killzone". The *Gallowdark* ruleset instead uses a 27″ by 24″ board with the specialised terrain shown in Figure 1. As stated previously, the terrain we are focusing on here are the thin walls with pillars connecting them. The terrain is all at the same height, but the operatives are shorter than the terrain. As a result, the system will need to find a solution to detect operatives behind terrain. From a birds eye view, the terrain will block part of the operative due to parallax. For this project, we will focus on using a half sized board to simplify this problem.

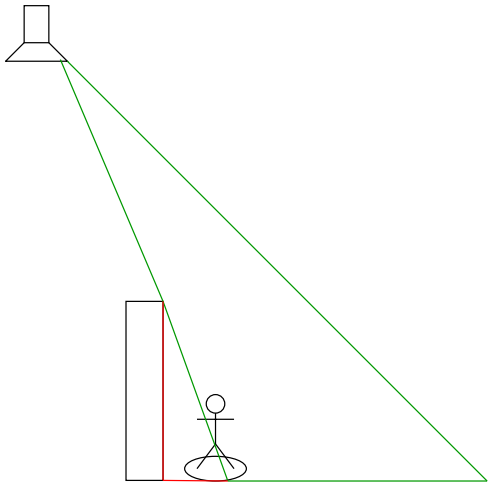

Figure 3: An example of the problem introduced by parallax. The camera is placed above the board offset from the operative. The operative is partially obstructed by terrain. The green triangle represents the parts visible to the camera. The red lines demonstrate the areas blocked by the terrain.

The further away the camera is from the operative on the x axis, the more the terrain will block the operative. However, as the camera travels away on the y axis, the terrain will block less of the operative. For a camera implementation this would also mean we lose quality in the image. As a result, a camera based system would need to find an ideal distance from the board which provides enough quality in the image, whilst also being able to see enough of an operative if it is obscured by terrain.

TODO

Put an IRL image in here to demostrate

2.1.2. Gameplay

The gameplay of Kill Team focuses around a turn based system with a full game consisting of 4 rounds - called "turning points". Each player takes turns "activating" a single operative and performing actions with it. The number of actions available to an operative is defined by it's statistics[1]. Once an operative has performed it's maximum number of actions, play is handed to the other player. That same operative may not be "activated" until every other operative, still in play, on their team has been activated. Once every operative on both teams has been activated the next turning point begins resetting the activation of each operative still in play. Once all 4 turning points have been completed the game ends and a winner is decided.

This project is focussed on creating a digital representation of the board, and then implementing a few select rules from the game to demonstrate the system's capabilities. The rules for "movement" and "shooting" present themselves as good candidates for this due to both their complexity and frequency within the game.

Please note that the explanations for these rules are abstracted from the *Kill Team Lite* rules published publicly by Games Workshop [3].

2.1.2.1. Movement

There are two types of movement available: "normal move" and "dash". A normal move allows an operative to move a set distance as defined in it's statistics (typically this is 6″). A dash allows an operative to only move 3″. Movement has several restrictions attatched.

1. Movement must not exceed the operative's movement characteristic.
2. Movement must be made in straight line increments. This means no curves but diagonals are fine.
   a. Increments can be less than 1″ but are still treated as 1″ for the purpose of total movement.
3. An operative cannot move through terrain unless it is a "small obstacle" (such as a barricade)[2]
4. An operative may not move over the edge of the board or through any part of another operative's base.
5. An operative CAN perform a normal move and dash in the same turn.
6. An operative CAN perform a normal move OR dash and then perform a "shoot" action.

This is quite a complex set of rules to enforce, making movement a good candidate for the system to assist with.

TODO

Maybe add some images to demonstrate

2.1.2.2. Shooting and Line of Sight

Operatives can be set to two states: "concealed" or "engaged". These states are used to determine whether an operative is a valid target or whether it is able to make offensive actions, such as shooting. An operatives state is denoted by a small triangle of orange card pointing to the model. The system will need a way to track the state of an operative, whether this be through detecting the orange markers or manually updating the state.

[1] There are a lot of exceptions to this with abilities being able to modify the stats of themselves or other operatives, but for the sake of simplicity we are going to ignore this and assume stats are set.

[2] We will not be encountering small obstacles in this implementation.

Anecdotally, the *Kill Team* line of sight rules tend to be the most complex part of the game and are constructed in such a way that experienced players can abuse them to gain an advantage such as one way shooting[3].

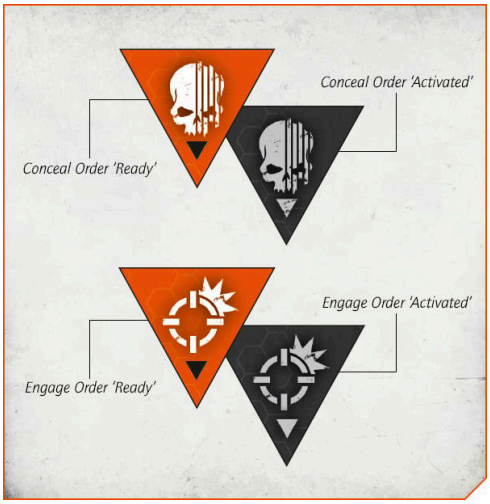

Figure 4: An example of the markers used to denote the state of an operative [7]. Once an operative has been activated the marker is flipped.

There are a set of requirements that must be met to determine whether a target is able to targeted by a shooting attack. We will use **attacker** to refer to the operative making the attack and **defender** to refer to the intended target.

For a defender in the engaged state:

1. The defender must be **visible**.
2. The defender must be not **obscured**.

For a defender in the concealed state:

1. The defender must be **visible**.
2. The defender must be not **obscured**.
3. The defender must be not **in cover**.

Kill Team defines **visible**, **obscured** and **in cover** very strictly. This is done using **cover / visibility lines** which are straight lines 1mm wide drawn from one point on an operative to another operative.

2.1.2.2.1. Visible

Visible is used to simulate whether a defender can be physically seen by an attacker

For a defender to be (visible) the following must be true:

1. A visibility line can be drawn from the head of the attacker to any part of the defenders **model**, not including the base.

2.1.2.2.2. Obscured

Obscured is used to simulate whether a defender is blocked by large terrain, such as a wall, while also containing edge cases for operatives peeking round said terrain or attempting to use it directly as cover.

[3]This is a technique used to allow an attacker to be able to select a valid target but the defender is unable to fire back.

When drawing cover lines the attacker picks one point on their base and draws two cover lines from that point to every point on the defenders base, to create a cone. In practice this means drawing two cover lines to the extremes of the defenders base.

TODO

Going to need to put some examples in - maybe use the pre-existing ones? not sure if this is ok to use: https://www.reddit.com/r/killteam/comments/vukgpz/basic_line_of_sight_rule_slate_i_made_for_our/#lightbox

For a defender to be **obscured** the following must be true:

1. The defender is >2″ from a point where a cover line crosses a terrain feature which is considered obscuring[4].
   a. If the attacker is <1″ from a point in which a cover line crosses a terrain feature, then that part of the feature is not obscuring.

2.1.2.2.3. In Cover

In cover simulates whether an operative is attempting to intentionally take cover behind piece of terrain. A concealed operative is hiding behind the terrain whereas an engaged operative is poking around / over it.

For a defender to be **in cover** the following must be true:

1. The defender is >2″ from the attacker
2. The defender is <1″ from a point where a cover line crosses a terrain feature.

There are a few key takeaways from these rules.

1. It is important to know the positions of operatives and terrain.
2. The system will need to be able to know the size of an operative's base.
3. Rotation of operatives is not needed to be tracked (except for visiblity when determining **visible**).
4. Operatives need to be marked as concealed or engaged.
5. Accuracy is important to the system. Visiblity / cover lines being 1mm wide and distances measured in inches will require accurate measurements.
6. The system will need to asbtract the above rules whilst also being able to display the reasoning behind the application. For example, if a defender is obscured the system should show what is obscuring and from what point the firing cone was drawn.

2.1.3. Community

Tabletop wargames require you to build and paint your own minitatures, because of this the target audience tends to be more of the creative type. Using this information we can allow some liberty in what they might need to do to utilise this system. For example, creating homemade tags is something that creatives might be more inclined to do as opposed to something more technical using RFID or infrared sensors. The whole system should be designed to be as accessible as possible to the target audience, requiring little to no specialised equipment.

The minitatures are also not defined in their appearance. They can be painted in any scheme, be customised to have completely different looks or use entirely different models than what is intended[5].

[4]For our use case the terrain walls in Gallowdark are all considered obscuring terrain.

[5]Known as proxying.

This means the tracking system needs to be ambivolent of what the model looks like, both in colour and silhouette.

Since miniatures are highly customisable players tend to be very attatched to them. So one important requirement is to not be invasive to the miniature. This rules out requiring a certain paintjob, placing stickers on heads or putting QR codes above operatives. The system should aim to obscure models as little as possible and cause no damage.

3. Related Work

3.1. Previous Mixed Reality Tabletop Systems

Some companies, such as *The Last Game Board* [8] and *Teburu* [9], sell specialist game boards to provide a mixed reality experience.

The Last Game Board achieves this through utilizing the touch screen to recognise specific shapes on the bottom of miniatures to determine location and identity. *The Last Gameboard* is 17″ x 17″, as a result the number of game systems which are compatible is limited. However, you can connect multiple systems together. The drawback of this is the price point for the system is rather high, with boards starting at ~$350. It is also worth noting that this system has not recieved great reviews, with alpha users reporting: long load times, low FPS, graphical and sound glitches, lack of updates and even screens refusing to display half the screen for certain applications.

Figure 5: The Last Game Board touchscreen tabletop system [8]

Teburu [9] instead takes an RFID based approach, providing a base mat that allows you to connect squares containing RFID receivers and game pieces containing an RFID chip. *Teburu* connects to a tablet device to provide the digital experience as well as to multiple devices for individual player information. *Teburu* games allow for game pieces to either be in predetermined positions or within a vague area i.e. within a room.

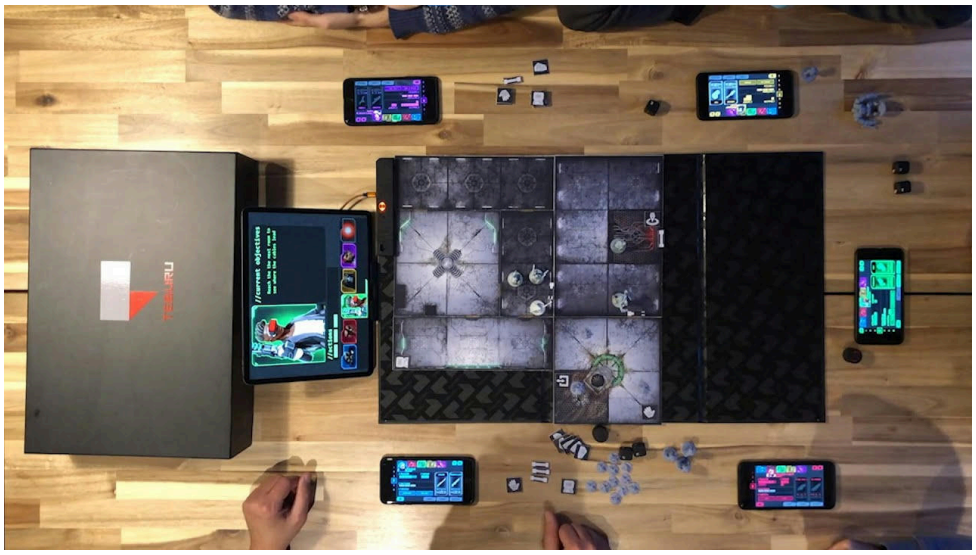

Figure 6: The Teburu Game System [10] showcasing *The Bad Karmas* board game. The black board is the main game board. The squares above connect to the board below to transmit the RFID reader information back to the system for display.

An RFID based approach is also used by Steve Hinske and Marc Langheinrich in their paper [11] which places an antenna grid below the game board to detect RFID chips within models. This allowed them to find what chip is in range of what antenna, allowing them to find the general location of a game piece. This worked particularly well for larger models where you could put RFID chips far away from each-other on the model. Using the known positions of the chips and dimensions of the model combined with which antenna said chips are in range of allows you to determine an accurate position of each model. They also go into alternate RFID approaches which will be discussed later when outlining the chosen methodology.

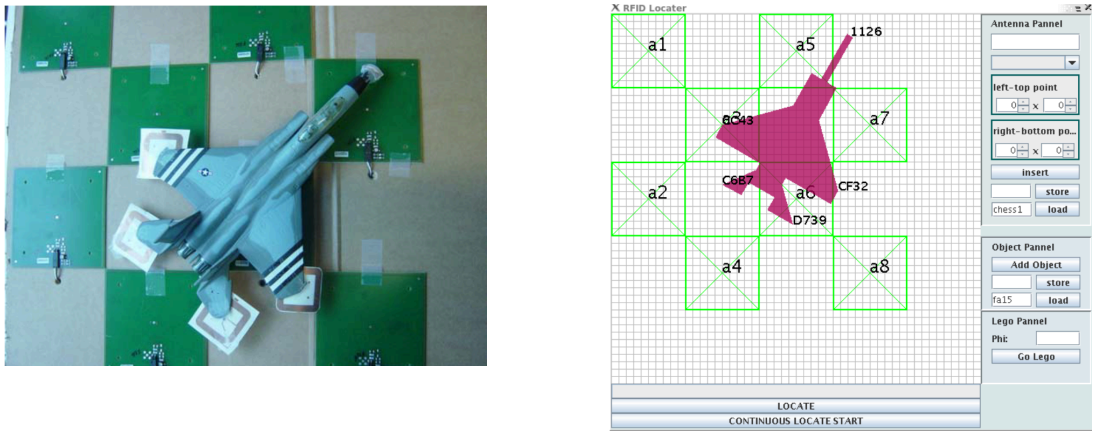

Figure 7: An example of Steve Hinske and Marc Langheinrich's approach depicting the antenna grid, RFID tags and physical model alongside the computer's prediction of the model's position

Surfacescapes [12] is a system developed in 2009 by a group of masters students at Carnegie Mellon as a university project. *Surfacescapes* uses a *Microsoft Surface Tabletop* (the product line was rebranded to *PixelSense* in 2011). This uses a rear projection display, and 5 near IR cameras behind the screen [13]. This allows the *PixelSense* to identify fingers, tags, and blobs touching the screen using the near IR image. *Surfacescapes* utilises this tag sensing technology to track game pieces using identifiable tags on the bases of miniatures.

Figure 8: An example of *surfacescapes*' in use on the *Microsoft Surface Tabletop*[14]. The position of the models have been tracked by the system and outlined with a green cricle.

Foundry Virtual Tabletop [15] is an application used to create fully digital *Dungeons and Dragons* tabletops. These can either be used for remote play or in person play using a horizontal TV as a game board. *Foundry VTT* allows for the creation of modules to add new functionality to your virtual tabletop. One such module is the *Material Plane* [16] module which allows the tracking of physical miniatures on a TV game board. This functions by placing each miniature on a base containing an IR LED with an IR sensor then placed above the board. This can be configured to either move the closest "virtual" model to where the IR LED is or (with some internal electronics in the bases) can be set up to flash the IR LED in a pattern to attach different bases to specific models. An indicator LED is present to show when the IR LED is active.

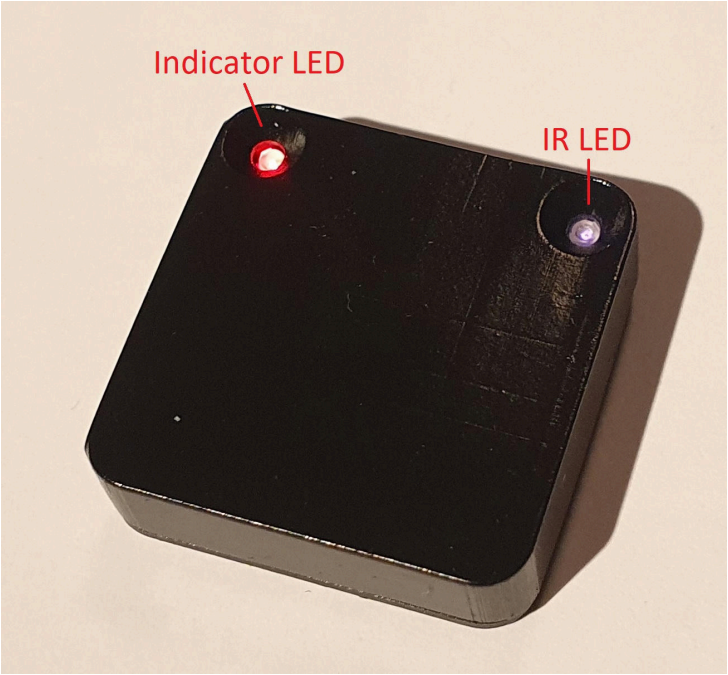

Figure 9: An example of one of the *Material Plane* bases. A miniature would be attached to the top. [17].

3.2. Object Detection and Tracking Technologies

Finding a method to detect and find the positions of miniatures on a game board, whilst not obstructing the game, is the main focus of this project. This section will outline the different technological approaches that could be taken to achieve this.

3.2.1. RFID

An RFID approach is the somewhat obvious solution. This would involve embedding RFID chips underneath the bases of the miniatures. Then some method of reading these chips would then need to be embedded either within or underneath the game board. There are a number of different approaches that could be taken to locating RFID chips which have been outlined by Steve Hinske and Marc Langheinrich [11] in their work on a similar project.

RFID solutions would require either an antenna grid underneath the game board or multiple individual RFID readers. This would be a viable option as hiding an antenna grid below a board is a relatively simple and unobstructive task. The same goes for hiding RFID readers beneath a table.

The main drawback of RFID is that a reader (at its core) can only detect if a chip is in range or not (referred to as "absence and presence" results). Due to this, some extra methodology would need to be implemented to determine the position of a chip.

One approach utilises increasing the range of an RFID reader to its maximum and then subsequently reducing the range repeatedly. Using the known ranges of the readers it would be possible to deduce which tags are within range of which reader - and subsequently deduce from which ranges it is detectable in the position of the RFID chip. This approach could in theory provide a reasonably accurate position of the RFID chip. However, this approach would take a longtime to update as each RFID reader would need to perform several read cycles. Combined with problems caused from interference between the chips / readers, the fact that being able to vary the signal strength of an RFID reader is not a common feature and the need for multiple RFID readers, this approach does not meet the requirements for this project.

Another approach utilises measuring the signal strength received from an RFID chip and estimating the distance from the reader to the chip, known as received signal strength indication (RSSI) . This approach is much quicker than the previous method needing only a single read cycle to determine distance. Most modern day RFID readers can report RF phase upon tag reading. However, current RSSI methods have an error range of ~60cm caused by noise data, false positives / negatives and NLOS (non-line of sight) [18]. This won't work for this project given than the board size is 70 x 60 cm.

Triliteration is a process in which multiple receivers use the time of arrival signal from a tag to determine it's position. This suffers from similar problems to other RFID methods in that it produces an area in which the tag could exist within - as opposed to it's exact position. Combined with the need for 3 RFID readers, this approach fails to be accessible to the target audience.

The approach previously mentioned in Steve Hinske and Marc Langheinrich's paper, which utilised an antenna grid below the game board, seemed promising.

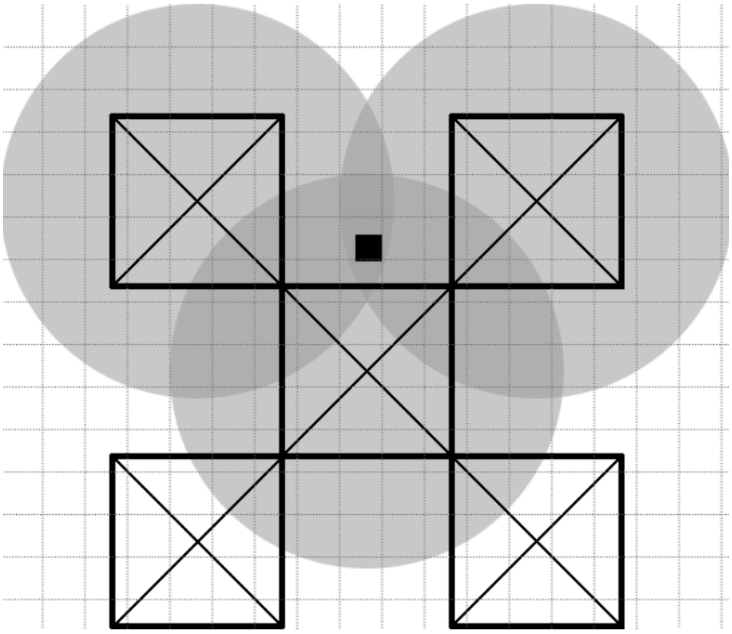

Figure 10: An example from Steve Hinske and Marc Langheinrich's paper [11] describing their overlapping approach. The black square represents the RFID tag whilst the grey circles show the read area of each RFID reader. The intent is to use which readers are in range of the tag to determine an "area of uncertainty" of the tag's location.

They were attempting to do this for *Warhammer 40k*, a game played on a much bigger board typically with larger models. As a result they were able to put RFID tags across larger models, such as vehicles, and utilise the known position of each tag relative to the model and the estimated position from the RFID readers to determine not only an accurate position but also orientation. Unfortunately for this project the size of the miniatures in *Kill Team* would require the use of one tag or two in very close proximity.

A potentially valid method utilizing RFID could be to use the approach outlined by Zhongqin Wang, Ning Ye, Reza Malekian, Fu Xiao, and Ruchuan Wan in their paper [19] using first-order taylor series approximation to achieve mm level accuracy called *TrackT*. However, this approach is highly complex and can only track a small amount of tags at a time.

3.2.2. Machine / Deep Learning

TODO

This should probably be improved upon - either by reading into deeplearning occlusion methods or recgonition methods of similar but different objects.

Modern object tracking systems are often based on a machine learning or deep learning approach. In this case a classifier model would be used to identify each unique miniature on a team, and then subsequently locate it within the game board. The biggest drawback to this approach is the amount of training data needed for each class in a classifier. According to *Darknet's* documentation (a framework for neural-networks such as YOLO) [20] each class should have 2000 training images.

Since each user will use their own miniatures, posed and painted in their own way, they would have to create their own dataset for their miniatures and train the model on them each time. As a user's miniatures are likely to follow a similar paint scheme, ML classification could potentially struggle to identify between miniatures from top down and far away if not enough training data is supplied.

3.2.3. ARKit

Apple's ARKit supports the scanning and detection of 3D objects [21] by finding 3D spatial features on a target. Currently any phone running IOS 12 or above is capable of utilising the ARKit. In this system, you could scan in your models, then use the ARKit to detect them from above. This information could then be conveyed to the main system. This could also allow for the system to be expanded in the future to use a side on camera as opposed to just top down detection, allowing for verticality within other *Kill Team* game systems. Combined with certain Apple products having an inbuilt LIDAR sensor (such as the *iPhone 12+ Pro* and *iPad Pro* 2020 - 2022), which further enhances the ARKit's detection, this could be a viable approach.

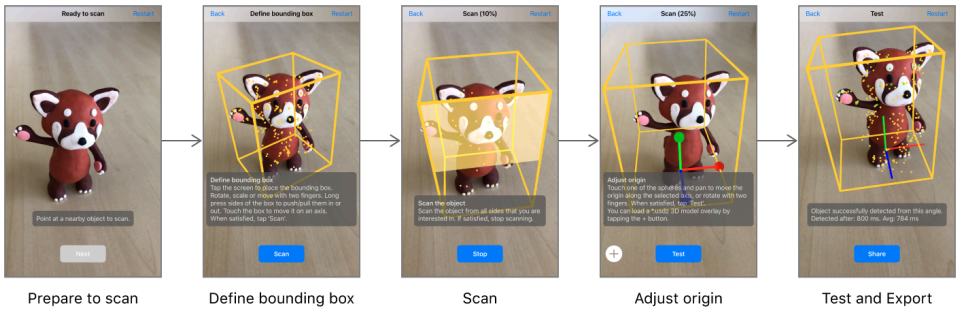

Figure 11: Registering an an object in ARKit [21]

After some testing utilizing an iPad Pro with a LIDAR scanner, some drawbacks were found with this approach. The object detection required you to be too close to the models detect them, and when being removed and re-added ARKit either took a long time to re-locate a model or failed to do so at all.

As a result I won't be using this approach for the project. Although, for future work, this could be an interesting option to explore allowing you to implement AR features such as highlighting models on your phone or tablet or displaying information above the model.

3.2.4. Computer Vision without neural networks

After considering all the options above, the best method found was using computer vision techniques.

One technique is blob detection. A blob is defined as a group of bright or dark pixels contrasting against a background. In this case the miniatures would be the blobs.

Utilizing blob analysis would allow locating general position of miniatures (as they would very clearly stick out from the background) but getting their exact center may then prove difficult (which would be needed for calculating movement and line of sight). Combined with the addition of terrain adding clutter to the image and the miniatures potentially being any colour, this approach could prove difficult to implement in the given time frame.

The computer vision method explored the most was utilizing coloured tags to identify miniatures.

The use of coloured tags would also mean there would be access to specific measurements allowing for an accurate location to be discerned. However, some external modification to the the game board or miniatures would need to be made. The challenge here is finding a way to do this without obstructing the flow of the game or the models themselves.

TODO

Actually talk about OpenCV

3.3. Computer Vision

TODO
Write up the papers on computer vision that were used.

3.3.1. Fiducial Markers and Tag Detection

3.3.2. Occlusion solutions

3.3.3. Object Detection

3.3.4. Parallax Solutions

3.3.5. Calibration Solutions

4. Project Description

This project aims to create a system that can track tabletop miniatures, in a game of *Kill Team Gallowdark*, using only materials easily accessible to the average miniature war-gamer. Then, utilise this system to create a digital representation of the game board. This digital representation will then be used to implement a few select game rules to demonstrate the system's capabilities and show that the tracking system provides the necessary information to process said rules.

The project can be broken down into two main goals.

1. Detection of the models and terrain to create a virtual representation of the game board.
   a. The position of the miniature must be tracked accurately.
   b. Be able to identify between different miniatures.
   c. Aim to be non-invasive to the miniatures.
   d. Be ambivalent to a model's shape and colour.
   e. Be able to complete this task whilst being accessible to the average miniature war-gamer. Not utilising any equipment the average wargamer would not have access to in their own home.
2. Implementing the game logic in the virtual board to guide players through the game.
   a. Allow users to select a model and which action they wish to preview.
   b. Calculate the distance a model can move and display this on the virtual board.
      a. Account for terrain that blocks movement.
   c. Calculate the line of sight between the selected model and opposing models then display this on the virtual board.
      a. Account for terrain that blocks line of sight.
      b. Display the reasoning behind the line of sight calculation.
      c. Display whether the target is obscured or in cover.
   d. Display information about the selected model's odds to hit a target.

The methodology chosen to achieve these goals must meet the takeaways from the project background section. These requirements are not included here as they are more implementation specific instead of outlining the functionality of the project.

TODO

I'm not entirely sure if this meets the aims of setting up the requirements. A large portion of this project was spent on researching the best methodology to actually use, so going in depth here on the specifics of implementation feels wrong.

5. Methodology and Design

5.1. Tracking Methodology

We settled on using a computer vision and tag based approach to detect the models and terrain. This would utilise placing a camera above the center of the gameboard on an adjustable stand and using tags to locate the models and terrain.

This approach was chosen for a few key reasons. Firstly, it is the most accessible option to the target audience. The only components needed are a camera, a computer, a printer and some method to position a camera above a board. Unlike RFID, IR or custom table methods, the average war-gamer would have access to these components.

Secondly, it is the most flexible option. A tag based system does not care about the shape or colour of the models, only the tags. This requirement meant that machine learning approaches would not be as viable as the potential variation in models used is too high. If we stuck to only supporting specific, unmodified models this may potentially work, but this would place a restriction on the target audience and in a subject such as miniature wargaming where customisation is so heavily valued, this would not be a good approach. As well as this given the unique paint schemes each player may use this would mean an ML model would need to be either trained specifically for each players kill team or be able to use a models silhouette to identify it. Both of these approaches either require users to make their own datasets and train the model themselves, a time consuming and technical task, or require a model that can identify a model from its silhouette, a complex task that would require a lot of training data and wouldn't be that accurate due to potential model customisations[6].

TODO

Some images demonstrating the same model with different paint schemes

Building on this, maintaining a unique identification for an object is a key requirement but is difficult to implement using a machine learning approach. Kill Team's can utilise multiple of the same type of operative, as a result the system would need to be able to track each "unique" operative despite having the same appearance. This becomes problematic when occlusion is introduced. If we were to move an operative and at the same time cover or move another operative of the same type, it would be difficult to differentiate which one was which. A potential solution would be to utilise similar technology to facial recognition models, particularly looking at research into identifying the differences between identical twins.

In contrast, a tag based system is ambivalent to the appearance of the model. This means the system is functional with any model, regardless of customisation which meets one of the main requirements of the project.

TODO

Actually put the citation in for this

TODO

Unsure if the footnote here is too informal.

[6]On a personal note, ai is not an area that I have much interest in and something which is, at the time of writing, currently all the craze. So doing something a bit different to the 'obvious' approach seemed interesting.

Finally, a tag based system will assist with accuracy. Getting the location and rotation of an aurco tag using pose estimation is a well documented process. This methodology can be applied to a tag design that will function well with miniatures and terrain.

TODO
Grab some citation for 'well documented process'

This project will use openCV [22] to process the video feed from the camera. This library contains great support for computer vision tasks and is available for both Python and C++. An alternative image processing suite would be *MATLAB* but this is not directly compatible with other languages so would require a lot of extra work to integrate with a visualisation system.

Python was chosen as the language for this project due to both prior knowledge and Python's loose type system contributing to the ability to produce quick working prototypes separate from the main project. C++ on the other hand would produce a more robust final product but would take longer to develop. As this project aims to serve as a proof of concept for the idea, Python fits the role better.

5.1.1. Camera Setup

A camera will be placed above the center of the gameboard looking downwards providing a view of the entire game board. The image will be distorted by two factors: Camera distortion and image distortion.

TODO
Get an image of the camera setup over the gameboard - or is that better to go in implementation?

Camera distortion is caused by different camera designs having slightly different distortion in the images they produce due to the different lenses. This can be corrected by using a camera calibration method to produce a camera matrix and distortion coefficients which can be used to un-distort resultant images. Radial distortions result in the image having straight lines appear curved. Tangential distortions result in the image appearing tilted along an axis. OpenCV provides a simple method to calibrate a camera using a chessboard pattern. This produces the camera matrix and distortion coefficients which can be saved and used to un-distort images. It is important to note that each different camera will have a different distortion so will need to be calibrated separately.

TODO
Image showing camera distortion from the openCV docs

Image distortion is caused by the location of the camera relative to the board. To correct this we can perform perspective correction. Taking a top down view of the board will not garuntee that the board will be rectangular in the image. For example, it is likey for the camera to not be perfectly level resulting in parts of the board appearing closer or longer than they actually are. To remedy this we can identify the four corners of the board and perform a perspective correction algorithm to produce a flattened version of the board. At the same time this will give us the boundaries of the board and crop the image.

TODO
Once again put an image in showing how this actually works - at the same time should probably cover aruco markers earlier

The board corners will be located with aruco tags [23] placed on the four corners.

TODO

Not sure if I leave this in cause I did not get around to doing it

TODO

Whilst this project is using a half sized board to simplify the problem it is important to note how a full sized board would be handled. Throw in an image to show how two cameras help with parallax - might just leave this out as it's not something I got round to doing.

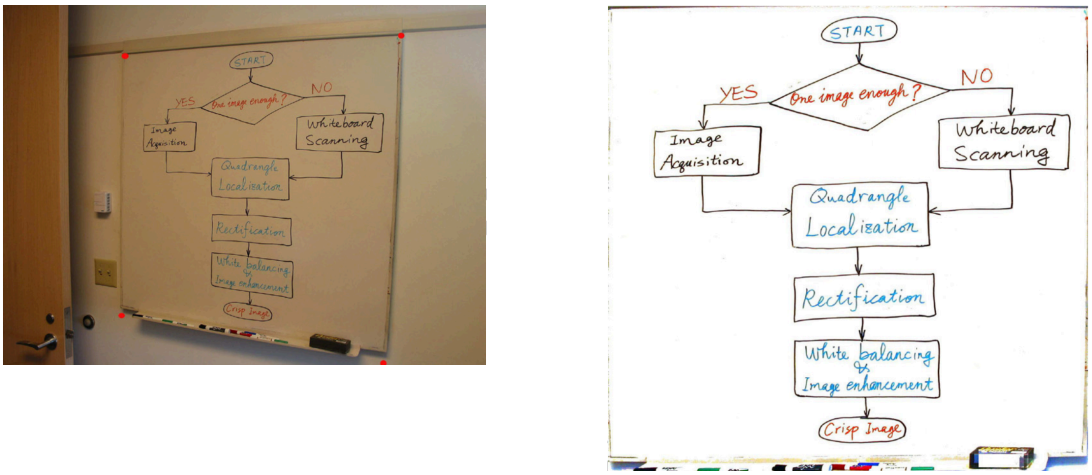

Figure 12: An example of perspective correction in Zhengyou Zhang and Li-Wei He [24]

5.1.2. Terrain Detection

Terrain detection utilises a straightforward approach. As the shape of terrain is limited in *Kill Team Gallowdark* to different constructions of pillar and wall. We can define a 2D model in the program to draw the terrain on the board. This means to detect terrain we only need to find a reference point to draw the model from, as opposed to needing to find the exact shape of the terrain.

This is achieved using aruco tags. As previously mentioned these allow us to perform pose estimation to find the location and rotation of the tag relative to the camera. Each type of terrain is defined by a unique aruco tag in a range. For example, pillar with one wall can sit in the id range 1-10, pillar with two walls in the id range 11-20 etc. Once we have identified the rotation, translation and scale of the tag we can apply these transformations to the model to draw the terrain on the board at the correct location.

The rotation of the aruco tag is provided in the form of a rodrigues rotation vector. However, our model library requires a angle rotation. To convert between the two we convert the rotation vector to a rotation matrix and then to euler angles. From here we can extract the z axis rotation (as we are working in a 2D plane from top down) and take the negative to get the correct angle for our model.

The terrain detection system returns a list of terrain objects with:
1. The position of the four tag corners
2. The rotation of the tag as an angle
3. The id of the tag

5.1.3. Operative Detection and Identification

Operative detection is more complicated. Unlike terrain we can't place aruco tags on top of models as this is obstructive to the game and would appear signficantly out of place. Instead we will design a tag to be placed around the base of the model.

The tag must be able to provide two pieces of information: the position of the operative and which operative it is.

5.1.3.1. Position

As the models are placed on top of circular bases the position of the operative can be defined by the center point of the circle and the radius of it's base. As the base size of a model is set, and the board size is also known, we do not have to worry about finding the radius of the base.

The requirements for the tag to provide the position are as such:

1. Be unobstructive to the model.
2. Be easily producable.
3. Be easily findable by the camera.
4. Provide the center point of the operatives base.
5. Achieve all of the above whilst also being able to be occluded by both terrain and the model itself.
    a. Ideally the center point should be able to be found even with a quarter of the tag being visible.

These requirements resulted in this tag design:

Figure 13: An example of the basic contrasting rim design.

To achieve this a high contrast rim will be placed around the base of the model. In this implemention we have chosen to use a bright yellow[7].

Utilising hough circle transforms [25] in openCV we can easily find the center point of a provided circle. The "completeness" of the curve required to be determed as a full circle can be easily adjusted to allow for occlusion.

[7] Although this could be changed to any colour which strongly contrasts the game board provided the correct thresholds were provided.

Before we can attempt to locate the center point of the circle we need to clear the image of noise. This is done by bluring the image[8], converting to HSV and then applying a threshold to only show the yellow colour space in the image. This leaves us with a binary image leaving only the yellow in the image.

TODO

Include images showing each step in this processing pipeline

From here we perform edge detection on the image to find the edges of the circles in the image. This is done as hough circle detection is a very computationally expensive process and by finding the edges of the circles first we are left with a wireframe of the circles which reduces the space the hough circle detection needs to search.

This leaves our processing pipeline as such:

1. Apply a gaussian blur to the image
2. Convert the image to HSV
3. Apply a threshold to only show the yellow colour space
4. Apply edge detection to the image
5. Apply hough circle detection to the image giving us a list of detected circle center points and their radii.

It is important to note that hough circle detection can easily mistake two close but separate circles as one circle.

TODO

A description of how HSV space works and why it's used here would be very useful

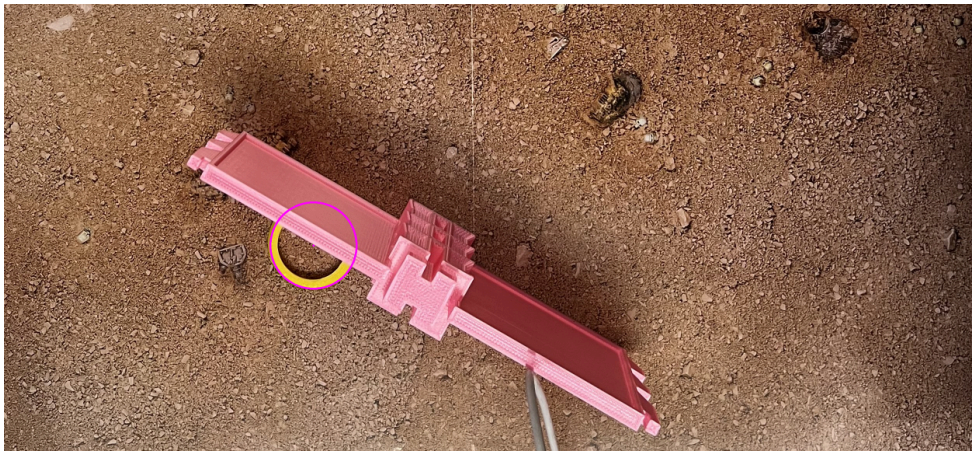

Figure 14: Circle detection on a gameboard with example terrain.

In extreme cases where the rim is heavily blocked by terrain the system will be unable to locate the model. However due to the nature of this project being aimed at following the *Kill Team Gallowdark* rule set, the terrain pieces used are placed along a grid. This prevents a situation where a terrain piece is very close to the edge of a board with a miniature placed behind resulting in the miniature being blocked from view.

[8] This is done to help reduce other noise from the image and soften edges.

5.1.3.2. Identification

The tag must also be able to provide a unique identifier for each model. The virtual gameboard will know which tag ID corresponds to which operative.

The method of identification on the tag must need to be functional whilst being occluded by terrain and the model itself. Ideally the tag should be able to be identified even if only a quarter of the tag is visible.

Kill Team's are typically made up of 7 - 14 operatives. This means our tag must be able to represent at least 28 different options.

To do this the following design was chosen:

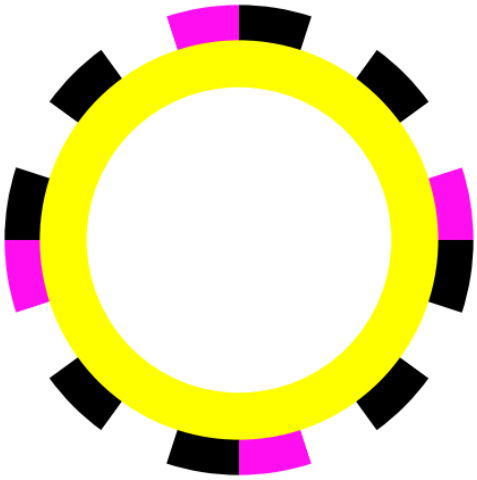

Figure 15: A tag representing the ID: 10. The outer ring is split into quarters with five sections within. The first section is a high constrast colour to indicate the start of the encoding. The four proceeding sections are split into black and white to represent the binary ID of the model. This pattern then repeats around the rim of the tag so only a portion of the tag needs to be visible to determine the ID.

Using 4 bits for identification allows for 16 different options. The first high contrast section colour can be changed to represent which team the model is on. This allows for 32 different options, more than enough for most Kill Team games.

The tag can be scaled to fit the size of the base of the model which is placed over the white circle. For this implementation we will use paper printed tags. This is a simple and cheap solution that is easy to get consistent colouring with. A tag of a similiar design could be 3D printed with indents to paint in the colours.

Having an outer rim provides two benefits. Firstly, it allows for the identification bits to be placed further away from the model, making it less likely to be obscured. Secondly, it acts as a barrier to prevent the yellow rim's from touching eachother. This is important as hough circle detection can easily mistake two close but separate circles as one, larger circle or multiple smaller circles.

The high contrasting starting bit (here in magenta) is used to determine the starting point of the encoding. From here the system can then read clockwise and anti-clockwise to determine the binary ID. It is important to note that the encoding uses big-endian encoding. When reading clockwise the first bit we encounter is the smallest. When reading anti-clockwise the first bit we encounter is the largest.

The process to get the encoding is as follows:

TODO
This should be a flowchart

TODO
Should also have a figure showing the resultant image at each step.

1. Take the circle centers and radii from the hough circle detection.
2. Using the same transformed image as before, but unblurred and unthresholded:
   a. Convert the image to HSV.
   b. Apply a gaussian blur.
   c. Apply a threshold to only show the magenta colour space.
   d. Perform an image dilation to ensure the magenta is a solid block.
   e. Find the contours of the magenta.
   f. Compute the centroids of the magenta.
   g. Find the four closest centroids to each circle center that are within the radius + a constant.

This will give us the starting positions of each visible encoding quarter for each circle.

From here we need to get the binary encoding.

The process to get the binary encoding is as follows:

1. For each circle:
   a. For each starting position provided (magenta centroid):
      a. Rotate the coordinates of the magenta centroid around the circle center to get the coordinates of each encoding bit.
      b. This is done both clockwise and anti-clockwise.
      c. Get the colour of the pixel at each encoding bit coordinate.
      d. If the resultant colour is within the threshold values for white, the bit is 1. If it is within the threshold values for black, the bit is 0. Anything else returns a NaN value.

This will give us a circle center, the associated magenta centroids and the clockwise and anticlockwise binary encoding associated with each magenta centroid.

Now that we have the colour values for each encoding bit in each quarter. We need to determine the final ID.

1. For each circle:
   a. Reverse the anti-clockwise readings.
   b. Group the associated enocdings bits lists by their positions in the encoding to create 4 lists (position 0, position 1, etc)[9].
   c. For each list:
      a. The final encoding bit for that position is the majority bit present.
   d. The final ID is the binary number created by the final encoding bits for each position.
   e. Create an object containing the circle center and the final ID.
   f. Add this object to a list of all the found circles
2. Return the list of all the found circles and their encodings.

TODO
Again this should be a flowchart

[9] for example 1 2 3 4 , 1 2 3 4, 1 2 2 3, 1 2 3 4 would become: [1,1,1,1] [2,2,2,2] [3,3,2,3] [4,4,3,4]

5.2. Game Board Representation

The interface is created using *Pygame*. This is a simple to use library that allows for the creation of 2D games. It is relatively lightweight and quick to develop with.

TODO

pygame citation

The digital game board represents a 22″ x 15″ board. Which is half the size of a standard board.

5.2.1. Terrain and Operative Placement

The size of the image will likely be different to the size of the digital gameboard. Due to this we need to scale the terrain and operative positions, provided by the tracking system, to match the scale of the gameboard.

In the case of operatives, the center point of the circle is scaled to match the gameboard. As the base size of the model is known, we draw a circle of the correct sized at the scaled center point. This keeps our digital representation accurate to the rules representation.

Terrain is slightly more complicated as it is defined as a series of 2D points to form a polygon. The terrain model is represented in mm sizes. The gameboard uses 3 pixels to represent 1mm. So we scale the terrain by 3x to match. We then rotate the terrain model to match the rotation of the tag

5.2.2. Line of Sight

5.2.3. Movement?

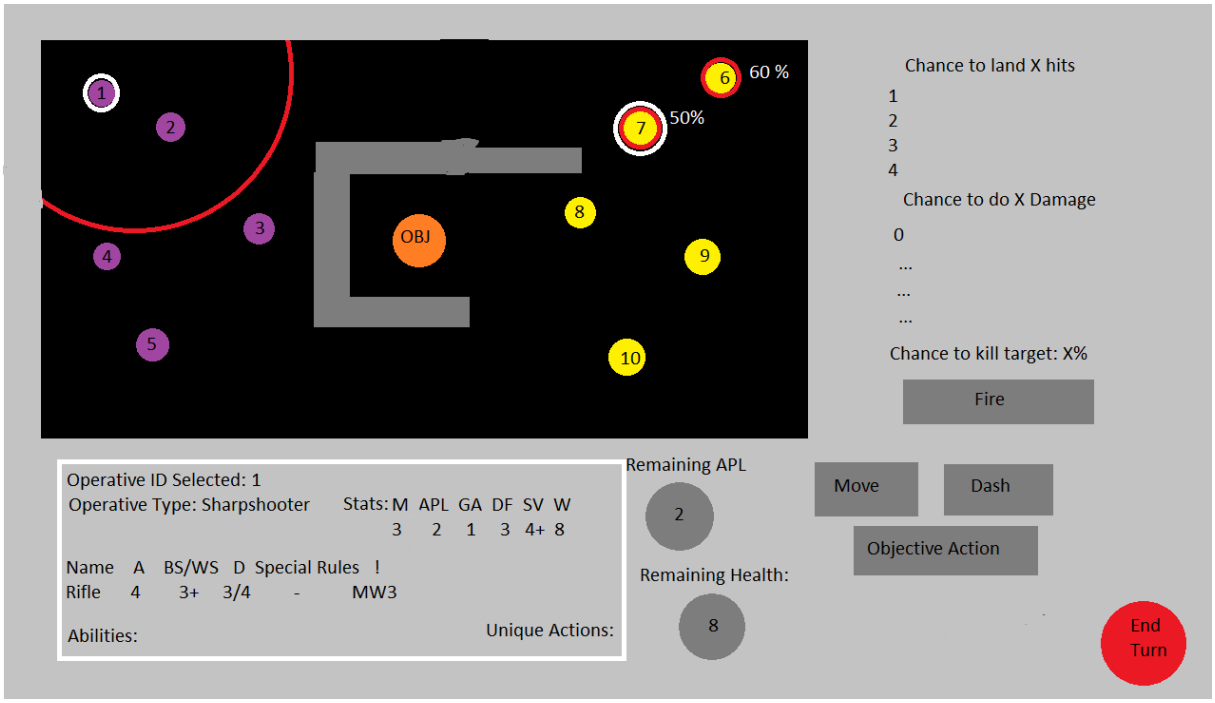

Figure 16: A proposed GUI for the game helper.

In Figure 16 the red semi-circle represents the valid area of movement for a selected operative. This can be calculated by splitting the game board into a very small grid, we can then apply a path finding algorithm (Dijkstra's, A* etc) to the grid from the operative's starting point. We can then calculate

all grid positions reachable by using a movement value less than or equal to the current operatives available movement characteristic.

Red circles around opposing operatives show they are in line of sight. This would be calculated using simple ray casting. A white and red circle indicates a selected opposing operative. This is to calculate the statistics on the right hand side of the screen to be specific to the opposing operative.

The information for the selected operative is displayed below the game board. A user can select the action they have performed / want to perform on the right hand side.

The GUI will be built in *PyQt* [26] which is a python wrapper for the *Qt* framework. One potential stretch goal is for the GUI and the detection system to be separate entities. For example, the detection system should be able to detect and track pieces without input needed from the virtual game board i.e. which model is selected. This would allow for other detection systems to be swapped out in the future or for other applications to be developed using the detection system.

6. Implementation

6.1. Camera Setup

6.1.1. Design and Calibration

6.1.2. Homography

6.2. Model Detection

6.2.1.1. Colour Thresholding

6.2.2. Rim Detection

6.2.2.1. Hough Circles

6.2.3. Model Identification

6.2.4. Optimisations

6.3. Terrain Detection

6.4. Game Board Representation

6.4.1. Linking of Detected models and terrain to the virtual board

6.4.1.1. Operatives

6.4.2. Line of Sight

6.4.2.1. Obsucring and visible

6.4.2.2. Cover

7. Evaluation

8. Summary and Reflections

8.1. Project Management

As per the initial Project Proposal the original goal for the first semester was to have ring detection and terrain detection completed. Currently, simple ring detection has been completed. This still needs to be expanded to include identification, though work has begun on this. Terrain detection has been moved backwards to allow for more time to make a good detection system for the miniatures.

Choosing to tackle the ring detection first was a good choice. This allowed for me to encounter larger issues and provide solutions to them whilst still in the research stages on the project. This has allowed for me to develop my computer vision skills and take approaches I would not have otherwise considered. For example, using a singular rim of multiple colours and combining the images to create a full circle for identification.

Supervisor meetings were held every week to check for blockers or advice on reports. Uniquely, our supervisor meetings were conducted as a group with the other two dissertation students my supervisor had. As all three of us were working on tabletop game based projects we were all knowledgeable in the area. This meant we could provide feedback or ideas for each others projects from a student perspective.

I found the Gantt chart to be unhelpful in managing the project. Gantt charts work well in well structured work environments. However, due to the nature of this semester having a large amount of other courseworks and commitments that needed to be balanced. An agile approach was much more effective, doing work when time was available. The nature of development work being much more effective when done in long, uninterrupted sessions meant that, with this semester's courseworks and lectures being very demanding at 70 credits, being able to find long swathes of uninterrupted time was very difficult between lectures, courseworks, tutorials etc. As a result, development was done in smaller, more frequent chunks which were not as effective as longer, less frequent chunks. I expect this to change next semester when the workload decreases down to 50 credits.

A much preferred method, that I will likely go ahead with, is Kanban. I personally found most use from the Gantt chart in that the project is naturally broken up into sub tasks. This approach of sub-tasking when, combined with a less structured agile approach, is something that Kanban works really well at and have found effective in the past in GRP (COMP2002).

In the interests of being able to show current progress in comparison to previous I have included an updated Gantt chart (Figure 17) along with the previous one (Figure 18).

Looking at the time remaining I will focus on getting the main parts of the system functional. These are: model detection and tracking, terrain detection, virtual game board representation, movement preview and line of sight preview. If there is time available then I will aim to also implement the flow of the game (breaking it down into each phase, providing guidance of what to do in each phase, statistics etc). The most technically interesting part of this is the virtual game board and tracking technology. So placing a focus on having them work fluidly is more important to the dissertation.

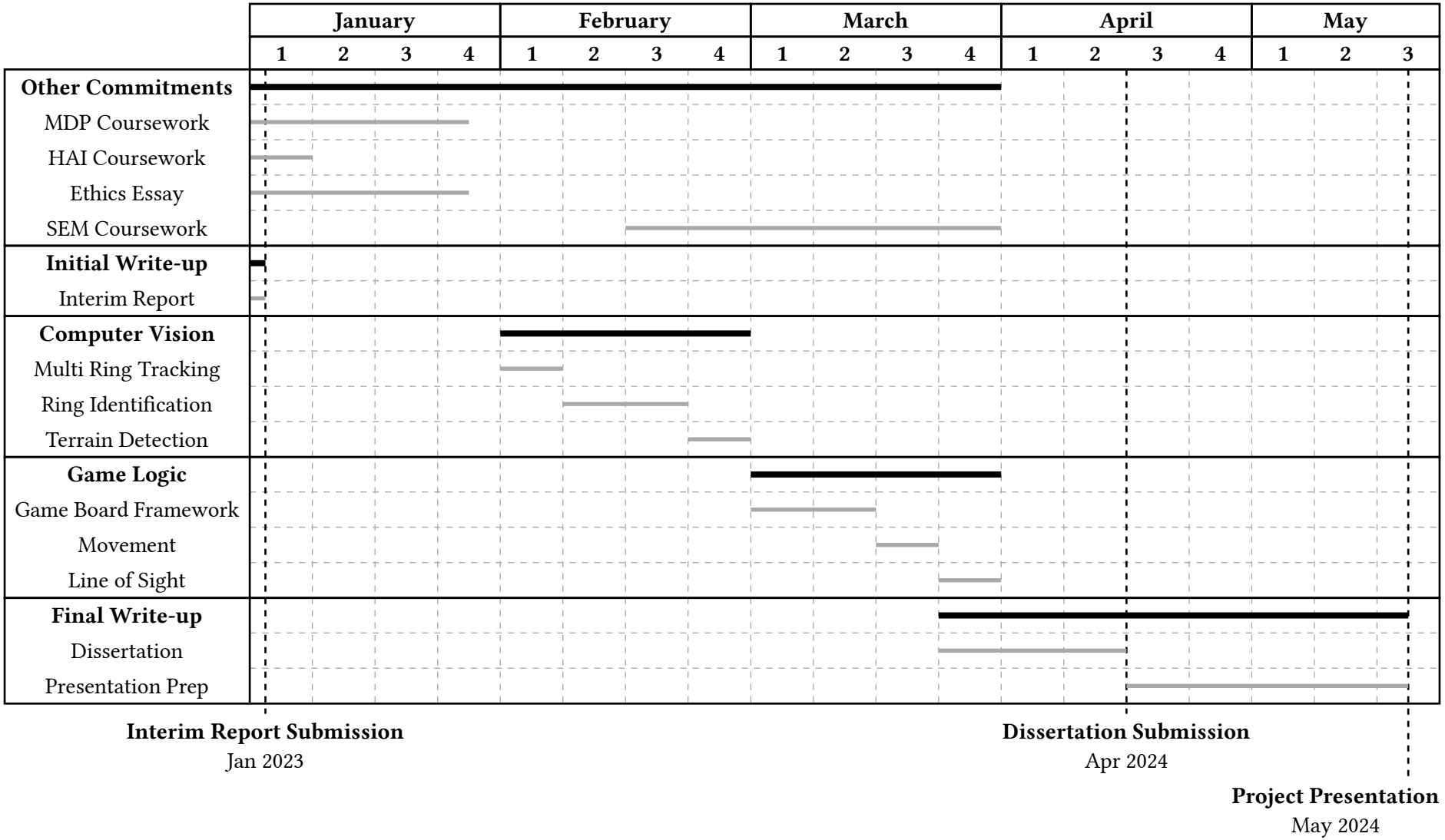

Figure 17: The Updated Gantt chart

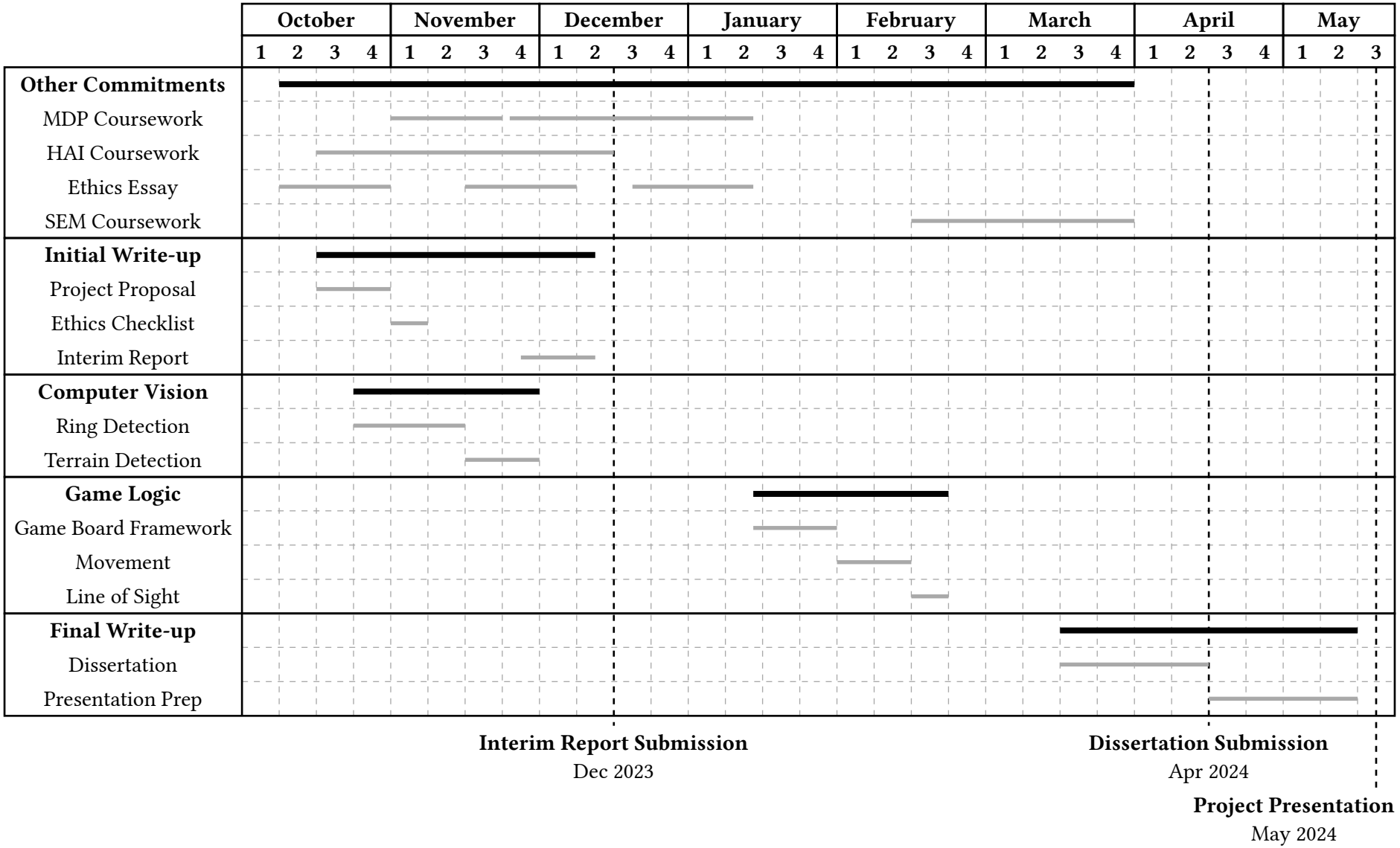

Figure 18: The original Gantt chart

8.2. Future Work and Reflections

Something I underestimated was the amount of time needed to research and determine which method should be used for model detection. A large amount of time was spent determining the viability of different approaches that may make the system more robust and easier to produce. As a result of this, development started later and took longer than was expected.

The biggest impact on this project was needing to get an extension on other coursework deadlines. This extension had a knock on effect of other courseworks which took priority over this project. The timings allotted in the Gantt chart did not take into account extensions being needed.

However, I am happy with the progress made so far. Having completed the methodology to find the circular tags even when partially obstructed is very promising to the viability of the chosen approach. As a result, I believe that a solid strategy has been chosen for model detection and identification and the work done so far has successfully laid a good foundation for the project.

8.2.1. Order Tokens and Objective Markers

8.2.2. Odds to Hit

8.2.3. Advanced Terrain

8.2.4. Height

8.2.5. Larger Game Board

8.2.6. Increasing the total number of operatives

8.2.7. Server Client Architecture For Game Board and Detection

8.2.8. Detection Upgrades

Hemming Distance

8.3. LSEPI

The main LSEPI issue you could see here is copyright issues as *Kill Team* is a copyrighted entity owned by *Games Workshop*. The way this project will handle this is by having the system implement the *Kill Team Lite* rule set previously mentioned. This is publicly available published by *Games Workshop*. One issue is that different "*Kill Teams*" (groups of operatives) will have different and unique rules as well as their own statistics pages. These are copyrighted and won't be able to be directly implemented. As a result of this, this project will implement basic operatives with fake statistic pages based off of the publicly published *Kill Team* information. If this proves to be problematic then the game rules will be based off a very similar system *Firefight* [27]. This is published by *One Page Rules*, who produce free to use, public miniature war-game systems.

8.3.1. Accessability

8.4. Final Reflections

Official openCV python documentation is somewhat lacking.

8.4.1. Design Approach

9. Bibliography

[1] Games Workshop, "Games Workshop Investor Relations Statement." Oct. 17, 2023. [Online]. Available: https://investor.games-workshop.com/our-history

[2] Games Workshop, "Warhammer 40k Core Rules." Jun. 02, 2022. [Online]. Available: https://www.warhammer-community.com/wp-content/uploads/2023/06/dLZIlatQJ3qOkGP7.pdf

[3] Games Workshop, "Kill Team Lite Rules." Aug. 16, 2022. [Online]. Available: https://www.warhammer-community.com/wp-content/uploads/2022/08/ekD0GG2pTHlYba0G.pdf

[4] Warhammer Community, "Kill Team: Into the Dark – What's in the Box?." Accessed: Dec. 31, 2023. [Online]. Available: https://www.warhammer-community.com/2022/08/04/kill-team-into-the-dark-whats-in-the-box/

[5] Games Workshop, "Killzone Gallowdark." Accessed: Dec. 12, 2023. [Online]. Available: https://www.warhammer.com/en-GB/shop/killzone-gallowdark-2023

[6] Games Worksshop, "Kill Team: Kasrkin." [Online]. Available: https://www.warhammer.com/en-GB/shop/kill-team-kasrkin-2023?queryID=4bbc70cecccea9006fc261421f8bc787

[7] Warhammer Community, "Sneaking and Spotting Are Just as Important as Shooting and Stabbing in the New Kill Team ." [Online]. Available: https://www.warhammer-community.com/2021/08/11/sneaking-and-spotting-are-just-as-important-as-shooting-and-stabbing-in-the-new-kill-team/

[8] Last Gameboard, "The Last Gameboard Kickstarter." Accessed: Dec. 12, 2023. [Online]. Available: https://www.kickstarter.com/projects/gameboard1/gameboard-1

[9] Teburu, "Teburu." Accessed: Dec. 12, 2023. [Online]. Available: https://teburu.net/#home

[10] "Teburu: a sneak peek." Accessed: Dec. 13, 2023. [Online]. Available: https://www.youtube.com/watch?v=5paNoKA4b5A

[11] Steve Hinske and Marc Langheinrich, "An RFID-based Infrastructure for Automatically Determining the Position and Orientation of Game Objects in Tabletop Games." Jun. 05, 2007. [Online]. Available: https://vs.inf.ethz.ch/publ/papers/hinske-pg07-rfidtabletop.pdf

[12] Whitney Babcock-McConnell, Michael Cole, Stephen Dewhurst, Bulut Karakaya, Dyala Kattan-Wright, and Maokai Xiao, "Surfacescapes." Accessed: Dec. 13, 2023. [Online]. Available: https://www.etc.cmu.edu/projects/surfacescapes/index.html

[13] Microsoft, "Microsoft Surface 1.0 documentation." Accessed: Dec. 13, 2023. [Online]. Available: https://social.technet.microsoft.com/wiki/contents/articles/8017.microsoft-surface-1-0-sp1-getting-started-guide-hardware-overview.aspx#Tabletop

[14] Abhay Buch, “With Surfacescapes, Dungeons & Dragons becomes high-tech.” Accessed: Dec. 13, 2023. [Online]. Available: http://thetartan.org/2010/4/5/scitech/surfacescapes

[15] Foundry Gaming LLC, “Foundry VTT.” Accessed: Dec. 23, 2023. [Online]. Available: https://foundryvtt.com/

[16] Material Foundry, “Material Plane.” Accessed: Dec. 23, 2023. [Online]. Available: https://www.materialfoundry.nl/

[17] Material Foundry, “Material Plane Github.” Accessed: Dec. 23, 2023. [Online]. Available: https://github.com/MaterialFoundry/MaterialPlane/wiki/Beta-Hardware-Guide

[18] “Robust and Accurate Indoor Localization Using Learning-Based Fusion of Wi-Fi RTT and RSSI.” 2022. Accessed: Dec. 27, 2023. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9002808/

[19] Zhongqin Wang, Ning Ye, Reza Malekian, Fu Xiao, and Ruchuan Wang, “TrackT Accurate tracking of RFID tags with mm-level accuracy using first-order taylor series approximation.” Dec. 15, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1570870516302736

[20] “YOLOv4 / Scaled-YOLOv4 / YOLO - Neural Networks for Object Detection (Windows and Linux version of Darknet).” Accessed: Dec. 28, 2023. [Online]. Available: https://github.com/AlexeyAB/darknet

[21] Apple, “ARKit.” Accessed: Dec. 27, 2023. [Online]. Available: https://developer.apple.com/documentation/arkit/arkit_in_ios/content_anchors/scanning_and_detecting_3d_objects

[22] G. Bradski, “The OpenCV Library.” 2000. Accessed: Dec. 12, 2023. [Online]. Available: https://github.com/opencv/opencv

[23] Garrido-Jurado S., Muñoz-Salinas R., Madrid-Cuevas F.J., and Marín-Jiménez M., “Automatic generation and detection of highly reliable fiducial markers under occlusion,” vol. 47, no. 6. Elsevier Science Inc. [Online]. Available: https://doi.org/10.1016/j.patcog.2014.01.005

[24] Zhengyou Zhang and Li-Wei He, “Whiteboard Scanning and Image Enhancement,” vol. 17, no. 2. Elsevier Science Inc., pp. 414–432, Jun. 15, 2006.

[25] G. Bradski, “Hough Circle Transform.” Accessed: Dec. 27, 2023. [Online]. Available: https://docs.opencv.org/4.x/da/d53/tutorial_py_houghcircles.html

[26] Riverbank Computing Limited, “PyQt.” Accessed: Dec. 31, 2023. [Online]. Available: https://riverbankcomputing.com/software/pyqt/intro

[27] One Page Rules, “Firefight.” Accessed: Dec. 31, 2023. [Online]. Available: https://www.onepagerules.com/games/grimdark-future-firefight