PathfindAR: An Augmented Reality App For Grid Navigation In TTRPGs

by

Rhys Agombar

Undergraduate Honours Thesis
Faculty of Science (Computer Science)
University of Ontario Institute of Technology

Supervisor(s): Dr. M. Green

Abstract

PathfindAR: An Augmented Reality App For Grid Navigation In TTRPGs

Rhys Agombar

Undergraduate Honours Thesis

Faculty of Science (Computer Science)

University of Ontario Institute of Technology

2017

Tabletop gaming sessions are time consuming endeavors since the math and strategy applied each turn needs to be calculated entirely by it's human players. Movement and attack ranges, areas of danger, and other pieces of grid information all need to be computed on the fly, which, when done by humans, is slow and prone to errors. To alleviate this we present 'PathfindAR', an augmented reality computer vision app that assists players with grid based navigation and range finding for tabletop role playing games. These activities are two of the most common and time consuming tasks that players do each turn, so by offloading these calculations to a computer, we can improve the speed of game play and make the overall experience more enjoyable.

Utilizing a combination of marker detection, contour detection, and positional tracking, we developed a portable system for android devices. The system computes and displays grid-related data by analyzing images taken from the device's camera and shading the affected grid squares with task-specific colours on the android device's screen. By allowing the android device to handle the tasks of range finding and navigation, this system allows players to make informed decisions on their turns quicker and lets them focus on the more fun aspects of strategy and game play rather than counting grid squares.

Contents

List of Figures

Chapter 1

Introduction

Modern types of tabletop role-playing games (TTRPGs) have existed in their current state since as early as the 1970s, with the first commercially available TTRPG, Dungeons & Dragons (D&D), being published in 1974 [8]. Since then, they have grown in popularity with numerous different systems and games being developed. Even with the advent and rise of video games as competitors, they still remain quite popular. Though the systems have evolved over the years, some core components remain mostly unchanged. Players typically represent their characters using figures or tokens, and use a grid to quantify things such as distances, terrain, character movements, and positions. Though some systems have moved away from the grid and instead prefer to deal with ranges and positions using abstract descriptors such as 'close range', 'medium range', and 'long range', rather than precise numerical values, it still remains a popular component in many modern TTRPGs.

Though computer technology has improved dramatically since the 1970s, TTRPGs have been slow to adopt it. Laptops have been used for storing character data and providing quick access to digital versions of the rule books and manuals, and projectors have been used to display animated or intricate maps on a table's surface, but very little has been done involving true augmented reality - something that has the potential to be

very useful in a TTRPG setting.

TTRPGs are algorithmic by their very nature. Almost everything is defined by structured sets of data and rules governing how things interact with each other. This makes computer assistance a viable possibility for augmentation.

1.1 Problem Statement

Various online tools have been created to assist with TTRPGs. There are online calculators for character statistics, generators for world creation, and macros for commonly used formulas from the game rules. What has not been touched, however, is the method of navigating using the grid. Grid navigation is one of the most time consuming and data heavy tasks conducted each turn. For example, in the Pathfinder or D&D game systems (the systems this app is designed to use), each turn a player needs to know: the ranges of their various forms of movement, the attack ranges of their weapons, areas of potential danger. In combat, this then needs to be compared with similar data from other players in the adventuring party, and against the enemies they are fighting in order to be able to plan effective strategies for the turn ahead.

Since remembering this much data is difficult, games often get slowed down by the grid navigation parts. Without computer assistance, players are forced to count squares and compute distances manually in order to plan their maneuvers - something which is time consuming even in the best case scenarios.

1.2 Goals

The objective of this thesis is to develop an augmented reality application to alleviate this problem by providing computer assistance to grid navigation and range finding. In order to be considered useful, the tool needs to be able to:

- Run on a mobile device and access a video feed from the mobile device's camera
- Identify a region in the video frames where a grid is found
- Identify human readable tokens and their positions within the grid
- Store token data relating to movement ranges, attack ranges, etc
- Display said ranges on a projection of the grid
- Display areas of danger through the use of blast/area templates

1.3 Thesis Outline

This thesis consists of six chapters, including Chapter 1, the introduction, which you are currently reading. In Chapter 2, we discuss PathfindAR's related work and similar software. In Chapter 3 we explain the design of the PathfindAR app, detailing its menus and how the information is displayed on the screen. In Chapter 4, we discuss the algorithms involved in implementing the computer vision system, as well as all of its interesting features and the tools used to accomplish this. In Chapter 5, we discuss the future work that could be done to extend and improve upon this system, and in Chapter 6, we conclude the paper and discuss how well we met our original project goals.

Chapter 2

Related Work

There are several works whose topics are related to PathfindAR. 'TARBoard' [6] was an attempt at augmenting tabletop games by using a glass table, a camera mounted under the table (looking up), and game pieces and cards with markers on the bottom. The system was not portable however, as it relied on a very specific hardware configuration and a glass table.

'Art of Defense: a collaborative handheld augmented reality board game' [5] was another project similar to this. In this case, it used a series of hexagonal tokens with markers on the top that could be identified by a camera phone and used to place turrets in a tower defense style AR game. The tokens however, were not human readable, though the system was portable.

'BattleBoard 3D' [3] was another game designed in a similar vein to 'Art of Defense'. It used AR markers to track the positions of game units and displayed them using AR goggles. Again, these markers were not human readable.

Lastly, the paper 'Augmented Reality RPG Card-based Game' [4] was very similar to what PathfindAR is intended to do. The game used a series of cards similar in design to trading card games like 'Magic: The Gathering', and tracked the cards as well as user interactions with them. It recorded things like the orientation of the card, the cards

attributes, name, ID, etc., and was able to use these to determine the game state: if a creature was attacking, defending, the damage done, etc. A computer monitor was used to display the AR overlay to the players, showing a 3D model of the creatures detailed in the cards and the damage numbers being calculated.

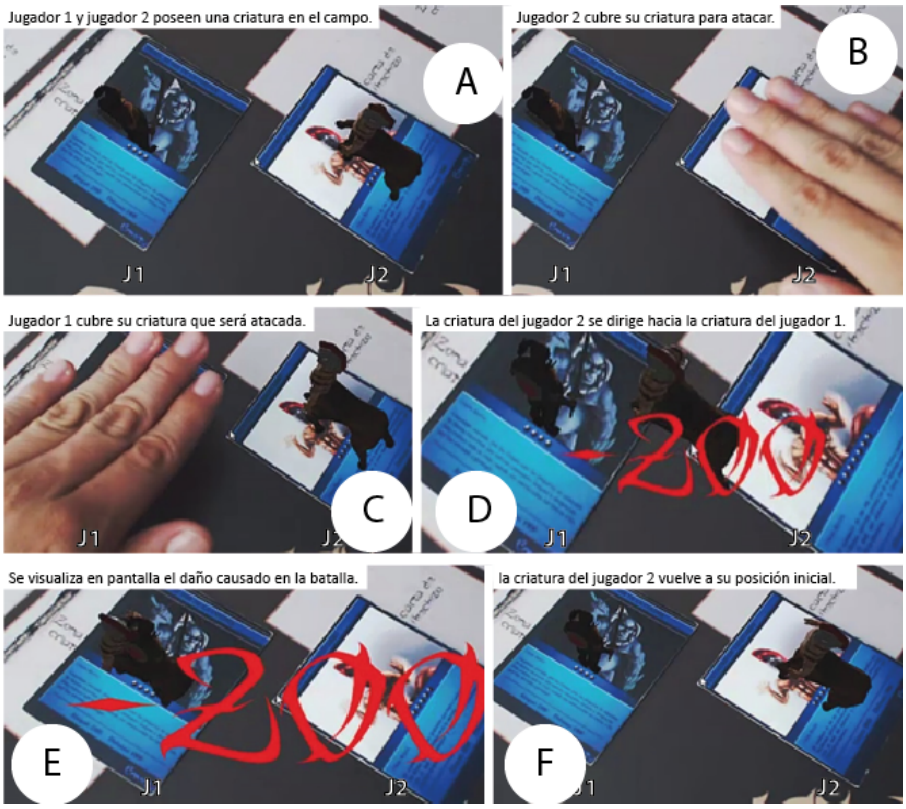

Figure 2.1: The card game in action. Figure taken from paper. [4]

While these applications are similar to PathfindAR at a high level, none of them meet the goals we have set out for this project. 'TARBoard' was not portable and relied on a very specific hardware setup for it to work. 'Art of Defense' and 'BattleBoard 3D' were both portable but the game pieces they were tracking were not human readable. 'Augmented Reality RPG Card-based Game' was the app that came the closest to PathfindAR, but also was not portable and was focusing more on augmenting the game with visual feedback like 3D models and damage numbers. Additionally, all of these approaches involved developing a game with AR in mind, rather than augmenting an

existing one. Compared to these, PathfindAR is different because of its objectives. It is designed to assist players by displaying important data from an already existing game system and help them plan their in-game actions, rather than displaying 3D models to improve game immersion. It is also designed to be portable, so it can be moved at a moments notice rather than being fixed in one place.

Chapter 3

Design

3.1 Main Screen

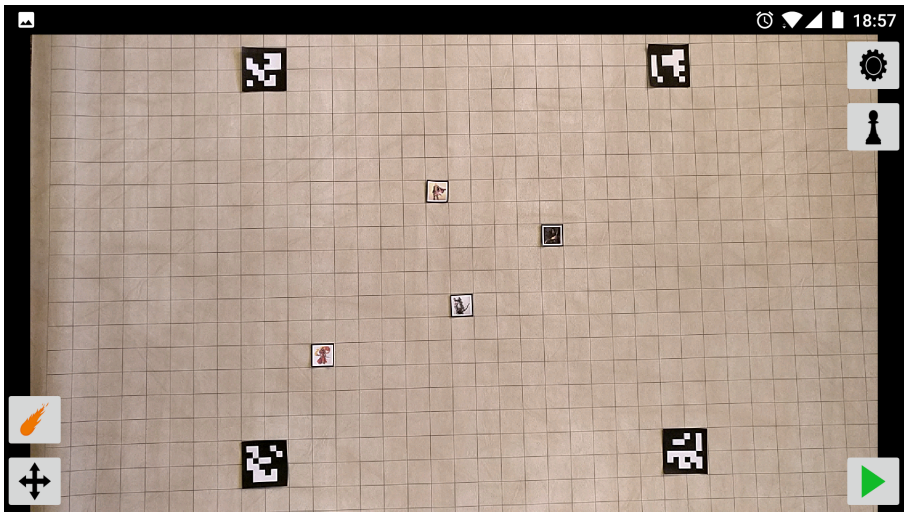

Figure 3.1: The app‘s main screen

Since PathfindAR is meant to be an augmented reality application, the main screen of the app shows the video captured by the device's camera. Depending on the options selected in the various sub-menus, the AR overlay will be drawn on screen here. In the upper left of the screen, an FPS counter is displayed. This is used to measure the performance of PathfindAR on the device. Due to the amount of data needed to be shown

on screen, the app is set to horizontal display mode. This provides more horizontal screen space for each entry in the text-heavy menus, discussed later in this chapter, and makes the app interface much less cramped.

In the lower-right corner of the screen is the tracking toggle button (Marked with ▶). This enables or disables the computer vision algorithms and AR overlay. When tracking is turned on, it draws a grid between ArUco markers found in the image and highlights any grid squares where tokens have been found, by filling them with an opaque, random colour. The grid dimensions are specified by the grid menu (discussed in the following paragraph), and the token data displayed is adjusted by both the token list menu and the toggle buttons located in the bottom left portion of the main screen.

In the upper-right corner of the screen, there are two buttons for the grid and token menus (Marked with ✹ and ♟, respectively). The grid menu button opens a new android view containing controls which are used for adjusting the dimensions (number of rows/columns) of the grid that has been or will be drawn onto the screen. The token menu button also opens another view, but in this case, it contains a list view that is populated by the tokens identified. This menu is used to select tokens in order to display their data on the main screen.

3.1.1 Grid data display modes

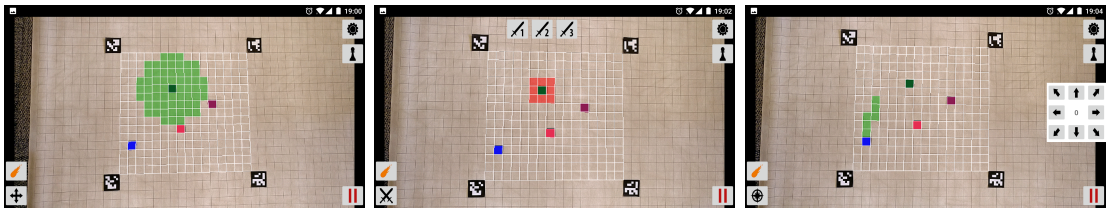

Figure 3.2: The three grid display modes: Movement, Attack, and Plotting

In the lower left corner of the screen, the buttons relating to data display modes are found. The lowest button is the grid data display button (Marked with ✢). Originally set

to movement, upon pressing it, it cycles between movement (✣), attack (✕) and plotting (⊕) modes. The movement mode shades the grid squares covered by the selected token's movement range with a partially transparent green, showing the area that a token can reach via standard movement.

When switched to attack mode, grid squares within the token's weapon range are shaded a partially transparent red. As per Pathfinder rules, reach weapons are unable to attack the squares within 5ft of the character using them, so if a weapon has the reach characteristic, the squares within 5ft of him will not be shaded to indicate this drawback. In addition, ranged weapons such as bows can make attacks between five and ten ranged increments away from the user, where each increment distance is the same as the base range. They are allowed to attack targets further away than their initial stated range, but for each additional increment of range they take penalties on the attempt. As such, this is key information for a player so when a weapon is designated as 'ranged', five different ranges are drawn in alternating shades of darker and lighter transparent red to indicate where these increment boundaries are. While it is true that some larger or more powerful ranged weapons in the Pathfinder and D&D rule set allow attacks at distances greater than five ranged increments, they also tend to have very long base ranges (+100ft). This means that drawing ten ranged increments for such a long ranged weapon is unnecessary, since the possibility of a user having a grid with more than 400 squares in length or width is very low.

When in attack mode, a set of buttons appear in the top center of the screen (Marked with 🗡). These buttons allow the user to switch between the different weapons their character is carrying to show all possible ranges that the character can reach with their attacks. The number of weapons that a character can be equipped with and cycled between has been capped at four. This follows the convention of most Pathfinder and D&D character sheets, which only have four weapon slots for a character. If a character has less than four weapons, the buttons not assigned to a weapon will not be visible.

The last mode is called plotting mode. At first glance, this does not display any data, but upon switching to it, a D-pad style button array appears on the right side of the screen, with the token's movement range displayed as a label in its center. By pressing these buttons, grid squares are shaded in the direction that the button press indicates, allowing the user to draw a path of movement for their character to follow. With each vertical or horizontal square moved, the token's remaining movement in the center of the D-pad is decreased by five. Pathfinder and D&D rules for diagonal movement are not quite that simple, however. In order to take into account the greater distance covered by diagonal moves, the rules state that every second square of diagonal movement costs 10ft of movement instead of the usual 5ft. As such, on the second, fourth, sixth, etc. presses of the buttons representing diagonal movement, the remaining movement label is decreased by ten instead of five. If a button press would cause a token to expend more movement than its movement range allows, the square is not colored and nothing happens. The plotted movement can be reset and the remaining movement returned to its original value by pressing the center of the D-pad.

3.1.2 Blast Templates

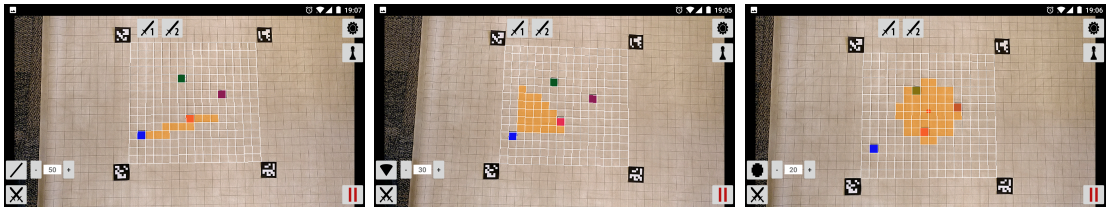

Figure 3.3: The three blast template types: Line, Cone, and Sphere

Located directly above the token data display button is the blast template button (Marked with). This button behaves in a similar manner, switching between 'line' (), 'cone' () and 'sphere' () modes on each button press and turns off the templates on the fourth press. When in line, cone or sphere mode, a pair of buttons and a label appear

beside it. These are used to adjust the length/size of the blast template in increments of 5ft. Since a large number of TTRPGs are set in high fantasy settings, this mode is meant to be used for calculating the ranges of magic spell or areas of danger. For the purposes of this app, we are using the Pathfinder rule set.

In line mode, the user first needs to have selected a token. They can then tap on a grid square and a line will be calculated between the selected token and the selected square. The grid squares along that line will be shaded with a partially transparent orange, up until the line length specified in the label beside the button has been reached. If the length is reached before selected square has been hit, the app will not shade any more of the squares and the remainder of the distance will be left uncoloured. If there is still distance remaining, more grid squares behind the selected location will be coloured, extrapolating from the line calculated in the initial selection phase until the full line length has been reached. The line drawn will be projected from the squares immediately adjacent or diagonal to the selected token, depending on the direction the line needs to travel.

In cone mode, like line mode, the user also needs to have selected a token to begin. They can then tap on a grid square and cone will be drawn outwards from one of the squares immediately diagonal, or two of the squares immediately adjacent to the player token. In the event that two squares are used, the position of the selected square is used to chose the position of the second square. The first one is always immediately adjacent in the direction selected position, and the second immediately diagonal to the selected token, on the angle that is closest to the selected point. The cone will then be projected out from these points. Like the line mode, the cone will stop short of the selected square if the cone size is not enough to reach it, or overshoot it if the cone size is greater than the distance to the square.

In sphere mode, the user does not need to select a token to project the template, because spherical spell effects typically do not originate from a token in Pathfinder.

Instead, they originate from a grid intersection (rather than a square), located anywhere on the grid. The user can select a grid intersection where ever they wish and the spherical blast template will be projected from it. Like the above templates, the labels and buttons shown beside the blast template button can be used to adjust its size

3.2 Token List Menu

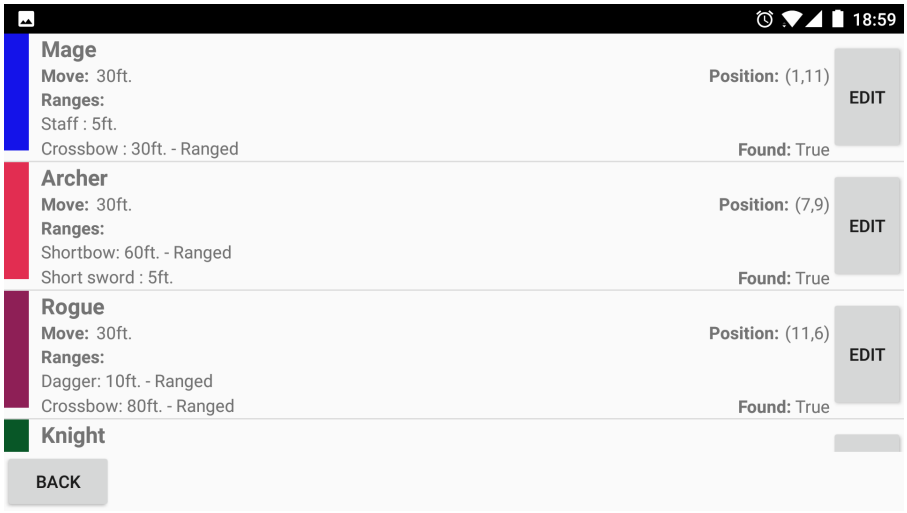

Figure 3.4: The token list menu

The token items in the list view contain: the colour assigned to them, their name, their position, if the token is still present or not, and an edit button to launch the token edit menu and enable changing the attributes. Since PathfindAR is designed with the Pathfinder and D&D systems in mind, the list also contains the token movement ranges, weapon attributes and attack ranges. Since Pathfinder and D&D represent each grid square as a $5ft^2$ region, all ranges are stated in 5ft increments. To select a token for display on the main screen, the user must simply tap the entry they wish to select. Tokens marked as found can be selected, but previously found tokens that are not visible in the screen cannot be. This selection immediately returns the user to the main screen to display the token data.

3.3 Token Edit Menu

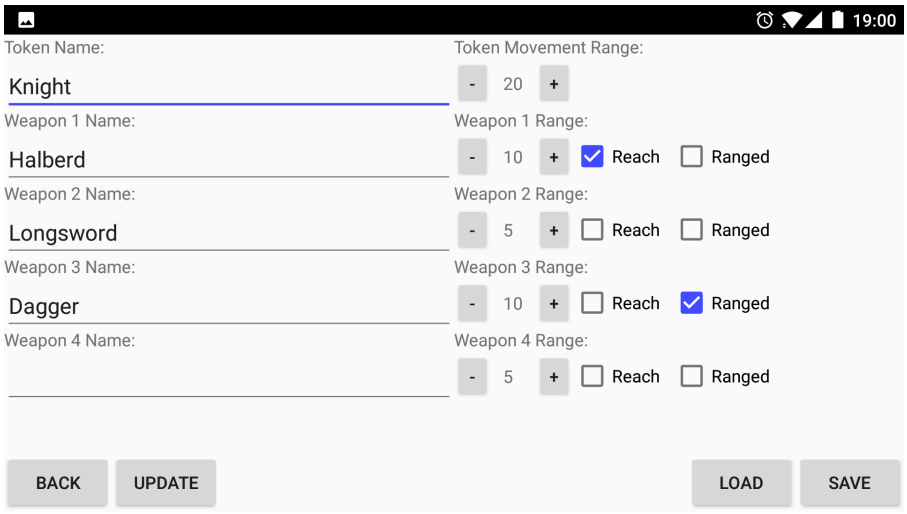

Figure 3.5: The token edit menu

The token edit menu, reached by pressing the edit button on a token's list entry, consists of a series of editable labels for the tokens and its weapon's names, as well as various checkboxes and buttons for altering the weapon attributes and ranges. The plus and minus buttons increase or decrease the range of their respective weapon in increments of 5ft, and the 'reach' and 'ranged' checkboxes allow their respective weapon attributes to be toggled between or turned off entirely. At the bottom left, the update button finalizes the changes made to the token and the back button returns the user to the token list menu, discarding the changes made if the update button was not pressed. On the bottom right side, the load and save buttons access a database of tokens. The save button saves the current token into the database for future use, and the load button opens the token database view, to select a token to load.

3.4 Token Database Menu

The token database menu displays a list of tokens similar to the token list menu, but with a few key differences: there is no colour assigned to them, there is no 'found' label, and

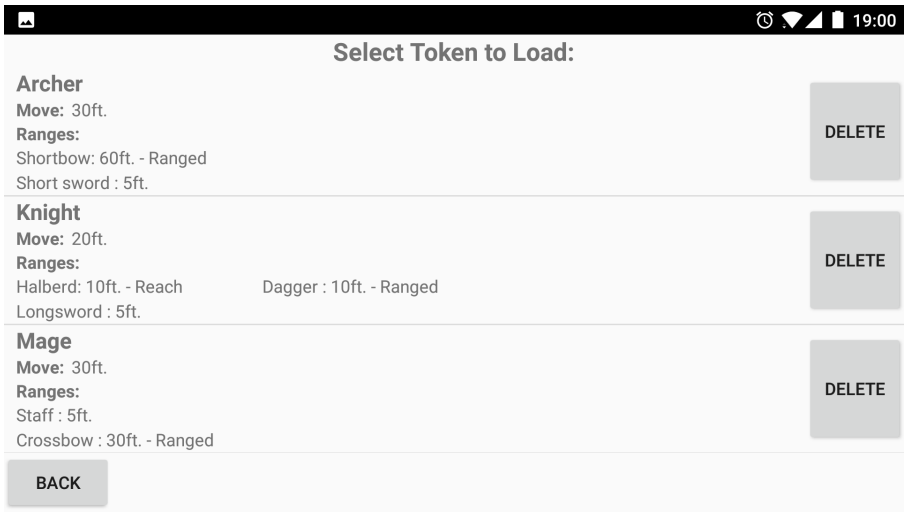

Figure 3.6: The token database menu

the edit button has been replaced with a delete button. By pressing the delete button, the token entry beside it is deleted and the list view is updated. By tapping the token entry (like in the token list menu), a token is selected and the user is returned to the token edit menu. The data from the loaded token is then used to populate the text boxes and labels, overwriting what was there originally.

3.5 Game Pieces

Figure 3.7: An example of the system's Token and ArUco markers

The game pieces used are a combination of ArUco markers and human readable images bordered with white and black stripes. The ArUco markers are used for marking the area the grid will be displayed in, due to their ease of detection and ability to be uniquely identified. The bordered images are used as game tokens to represent player character, monsters, etc. They consist of an image in the center to make them easily identifiable for humans, but are bordered with a white and black stripe to allow their edges to be detected easier. The game tokens are roughly 1" $\times$ 1", as 1" squares are the standard size of most TTRPG grids.

Chapter 4

Implementation

PathfindAR was developed for use with an android phone and the app was created targeting API levels 23-25, with testing being conducted on on Android Nougat v7.0 (API Level 24). The app uses the OpenCV4Android SDK (Version 3.1.0), OpenCV 3.1.0 with the ArUco module, and the Android NDK. OpenCV is an open source computer vision and machine learning software library [7]. In the context of this thesis, we are using it for all of the image analysis and computer vision related tasks. The ArUco module is an AR marker tracking library that is part of the OpenCV optional modules. It enables the use of ArUco markers, along with interpreting their encoded values and detecting their corners [2]. This is vital to the grid detection step, which will be described below. The Android NDK, or 'Native Development Kit' is a toolset that lets C and C++ code be executed on Android [1]. This was used to implement the computer vision algorithms needed, since the ArUco library only works in C++ and the algorithms perform better in C++ than in Java.

PathfindAR has been implemented in a combination of C++ and Java, with both languages being used for separate tasks. Java was used with the Android SDK to create the user interface, save token data, and accept user input. C++, using the Android NDK, handles the majority of the computer vision calculations, as well as drawing the

AR overlay. It is used for all of the time intensive calculations where speed is critical and packages are required that do not support android.

4.1 Grid Projection

When an image frame is captured by the Android device's camera, the java front-end code passes its RGBA components, as well as the desired grid dimensions, to the NDK's C++ code using the Java Native Interface (JNI). The C++ code then uses a combination of OpenCV and the ArUco library to detect the grid's location. The ArUco library searches the image for markers and returns a list of them, as well as their encoded values. These markers, once detected, are used to position the grid. Unfortunately, the ArUco library performs quite poorly with large sized images, reaching an unacceptably long processing time for each image frame. To mitigate this, we resize the image from 1280×720 to 720×480 and apply a gaussian blur in order to speed up processing time while still retaining accuracy and eliminating artifacts in the image.

For convenience sake, we use markers encoded with the values 0 to 3 for the corners of the grid, starting from the top left corner and counting upwards in clockwise fashion. When ArUco detects the markers in the image, it returns an array of the marker corner positions (labeled in similar fashion to the grid, 0 to 3 going clockwise), as well as the marker ID. We use the innermost corners of these markers as the corners of the grid to be drawn.

Unfortunately, due to the angle and position of the camera, if we were to draw the grid lines now, they would be inaccurate from the perspective distortion. As such, we need to draw the grid in an undistorted space and then apply a perspective transformation to warp it such that its projection matches what we would expect to see from the perspective of the camera.

To do this, we first transform the corners into a perfectly straight rectangle by finding

the maximum and minimum X and Y positions of the corners. We then adjust the corner positions, such that the top left corner has an X value equal to the X-minimum and a Y value equal to the Y-maximum, the top right corner has an X value equal to the X-maximum and a Y value equal to the Y-maximum, etc. The start and end positions of the grid lines to be drawn are then calculated on this straight rectangle, using the user specified grid dimensions to determine the number of horizontal or vertical lines and their spacing.

A homography matrix is then calculated from the four points of the distorted and undistorted grid corners. This is used to calculate the perspective transform of the grid line positions so that their positions remain accurate in the event that the camera is looking at the grid from a non-90° angle overhead. Once the grid has been drawn, it is indexed and each square is assigned X and Y values relative to the grid dimensions. The grid square data structures also keep track of the image coordinates of their corners for use in token detection.

4.2 Token Detection

Token detection takes place after the grid has been drawn and indexed. Once we know the locations of the grid squares (but before we actually draw them onto the map), we use contour detection to identify tokens and find their positions. In order to run the detection, first we blur the image with a 5×5 Gaussian kernel. This is done to reduce the possibility of detecting lines or marks on the physical grid that the tokens are on. Since the edges of the token are well defined with high contrast white and black stripes bordering the image, this does not negatively affect detection accuracy. After this, the detection function is run and a list of contours is returned.

The list is then filtered, with any contour that stretches outside of the grid area or is deemed too small being ignored. This prevents irrelevant edges in the image from

being misidentified as a potential token and cleans up any remaining contours from the physical grid that are still detected after the blurring. Once the filtering is finished, we then compute the centers of each contour and search the grid to find which grid squares contain them. This takes advantage of the fact that the square edges of the token all revolve around the center of the token, so even if the token is slightly offset from its grid square, the center point found should still be well within it.

Grid squares that contain at least two contour centers are considered to contain a token, and a list of unidentified tokens is then updated with the newly found token positions. The reason for this limit is because, despite the blurring and location filtering, noise or irrelevant marks on the grid still have the potential cause false contours to appear. This threshold, coupled with the previous measures taken, makes the token detection system very resilient to noise and false positives while still remaining accurate. Since the tokens in their worst case scenario have at least two contours and are much more likely to have three or four, tracking accuracy does not decrease.

4.3 Token Identification

The most straightforward way to identify tokens would be to use feature detection to both track and differentiate tokens from each other. Unfortunately, due to the small token sizes and video resolution, there is not enough data in the images to be able to do this accurately. Attempts were made, but were vastly inaccurate. As such, a different approach was required. Since there was insufficient data to use computer vision algorithms to solve this problem, we switched to using the token's positions to track them.

When the app first starts, we find every token on the grid and assign it a unique ID number (and record the token position for future reference). On every subsequent frame, the tokens found are compared to the previously saved list of identified tokens. For each

previously identified token, we check for:

1. Which unidentified token is closest to the current identified token
2. If any other identified tokens are closer to it
3. If another identified token is closer, we repeat with the second closest unidentified token for the same identified token

We then change the saved positions of the identified tokens to match the unidentified token they are 1) closest to, and 2) have no other identified tokens that would be a better match. Since token movement is a somewhat rare occurrence (relative to the frame rate), most tokens will remain in the same positions they were found at in the previous frame (thus making their distance 0, and guaranteeing they retain the same assigned ID). The few tokens that do move will have their IDs follow them. In the event of more tokens being found than previously identified, after we update the previous positions, the added tokens will be assigned their own IDs and added to the list of identified tokens. If a token is removed, it will not be found and the unassigned ID will be deleted from the token ID list.

This approach is not very robust, however. If a token is in the process of being moved between two grid squares, it will briefly be identified as two tokens and cause its ID to not follow it properly. Tokens can also be lost or added on occasion, due to sudden changes in lighting or noise in the image.

To compensate for this, we added a lifespan to the tokens. In order for a token to be identified, it must be present for at least seven frames before it is marked as a legitimate token. Once a token is identified, its lifespan is incremented on every frame where it is found, up to a maximum of twenty-five frames. If a token goes missing, its lifespan starts decreasing and until the missing token's lifespan counts down to zero, it is not deleted from the list. This prevents tokens from being lost due to a handful of bad frames and

makes tokens that have remained on the grid longer take more time to disappear. It also prevents a token from being duplicated or split when moving it between grid squares.

Chapter 5

Evaluation

In this chapter, we will demonstrate the app's capabilities by using it to play through a sample scenario. In this example, we have two factions that have begun to fight: a gang of ruffians (two rogues and an archer), and a pair of adventurers (a knight and a magician). The rogues and archer are positioned in the top half of the playing field and the knight and mage begin from the bottom. In this case, the playing area is a 10×10 grid.

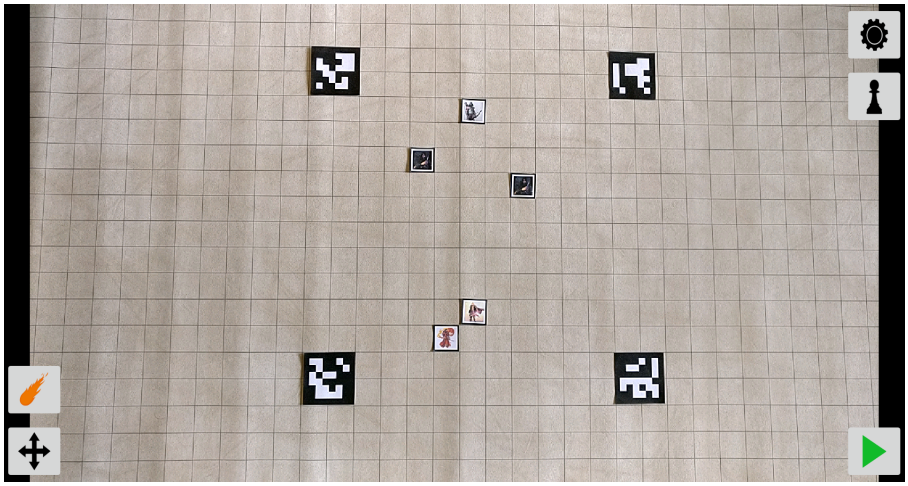

Figure 5.1: The starting positions of the tokens without an AR overlay

Before we can do anything here, we first need to set up the grid. The tokens are already in place, and the ArUco markers are positioned, so all that remains is to specify

the grid dimensions. We do this by entering the grid menu and adjusting the X and Y fields so that the dimensions are saved as 10×10.

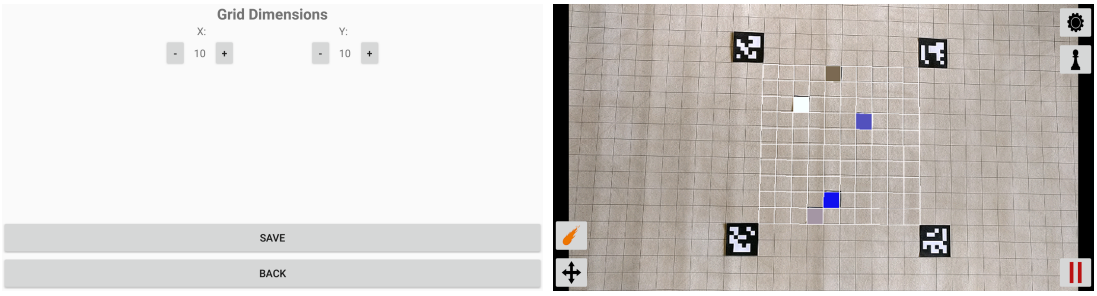

Figure 5.2: The grid menu and updated AR overlay

Once this has been completed, the grid is drawn and the tokens within it are identified. At this point, the token list is filled with placeholder data as the user has not entered anything into the system yet. Note that the colours on the left side of the list correspond to the colours of the tokens in the actual grid projection.

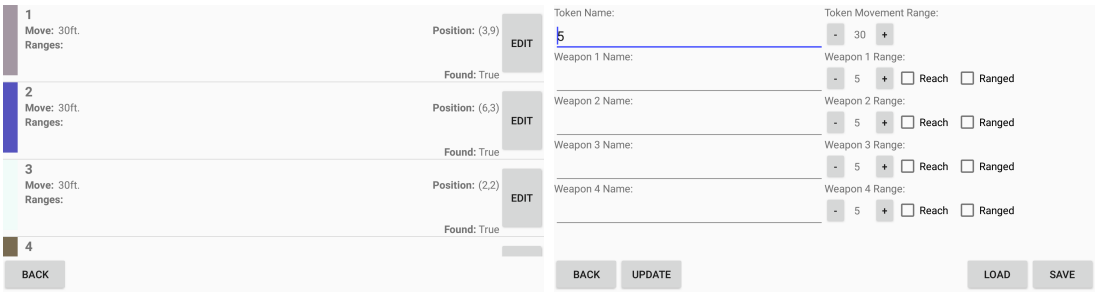

Figure 5.3: The empty token list and token data

This data can be input manually, but that has the potential to be time consuming. Fortunately, we have saved some commonly used token information into the token database that can be loaded. By utilizing the load menu, we can save time by simply selecting the token entry we wish to load and allowing it to update the token instead of typing everything out manually.

Once the everything is loaded, we can select the token we wish to use from the list and begin visualizing its data. In this scenario the knight (deep blue square), with his

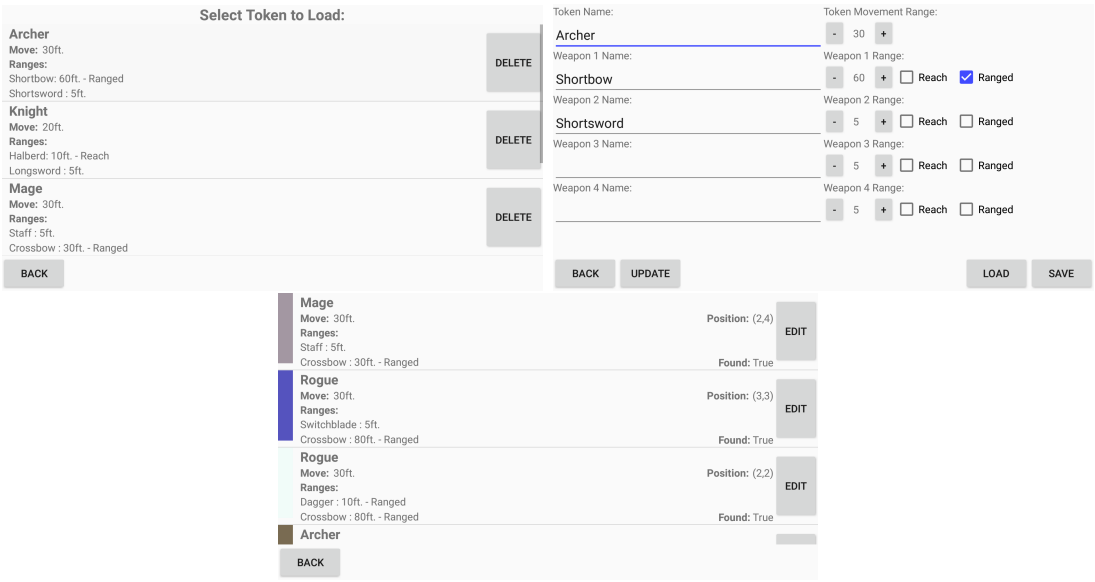

Figure 5.4: The load menu and filled token data, with updated list

familiarity with combat, has rolled the highest in initiative and will proceed first. He is going to attack the rogue on the left side (white square). The knight's primary weapon is a halberd, which happens to have a range of 10ft and is classified as a reach weapon (as seen in Figure 5.4). This means that he can attack at a distance but cannot hit enemies directly adjacent or diagonal to him. Additionally, due to his heavy armour, he can only move 20ft in a turn instead of the usual 30ft. In order to find out if he can actually get in range of the rogue, we turn on the movement display mode to show where he can move to.

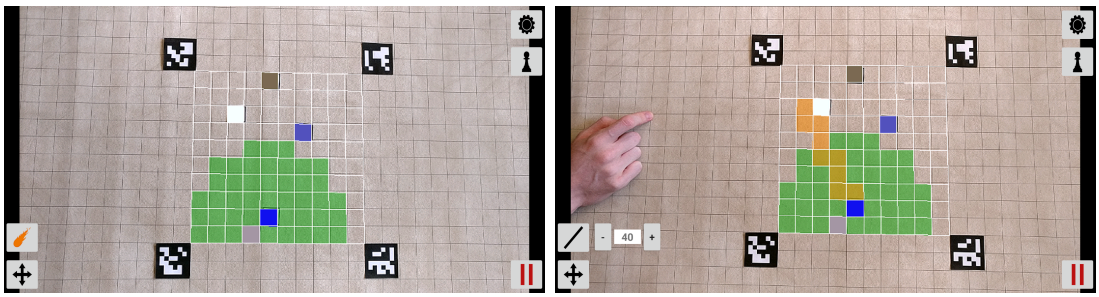

Figure 5.5: The knight's movement range and charge line

Unfortunately the knight, being weighed down by his heavy armour, is unable to get

into the position he wants (2 squares directly below the rogue) using standard movement. His movement range ends one square short. Fortunately, Pathfinder has a type of movement called a 'charge'. Charging allows the player to move their token up to twice their movement speed and get a bonus to attacking at the end, with one caveat: the movement must be in a single straight line. In order to determine if the knight can make this charge, we select the 'line' template and set its length to 40ft. We then selected several different squares, watching to see how the line formed, before settling on the line that allowed the knight to move into the best position. After this, we switch to attack mode (using the first weapon) to make sure our range is correct and have the knight perform its attack.

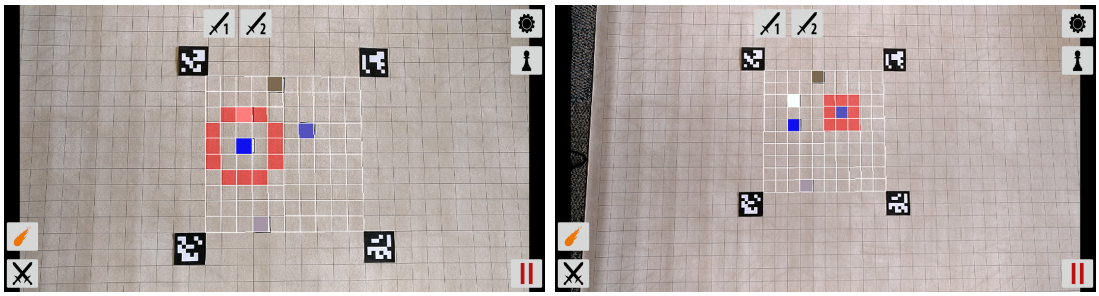

Figure 5.6: The knight's halberd range and the rogue's switchblade range

In this scenario, the next token to move is the rogue that is the furthest right (light purple square). As seen in Figure 5.6, the rogue only has a 5ft range compared to the knight's 10ft range. This means that if he were to directly approach the knight, he would have to pass through his attack range and possibly be hit in retaliation. Upon examining the knight's attack range however, we noticed gaps at each of the corners that would allow a character to move diagonally through them and avoid being hit. A strategy brought about by the limitations of using a grid for ranges, but a valid one none the less. In order to make sure that the rogue has enough movement range to perform this maneuver, we select the rogue's token and enter plotting mode.

As seen in Figure 5.7, the rogue has more than enough range to approach directly, and just enough range to move along the plotted course to close the distance safely. This

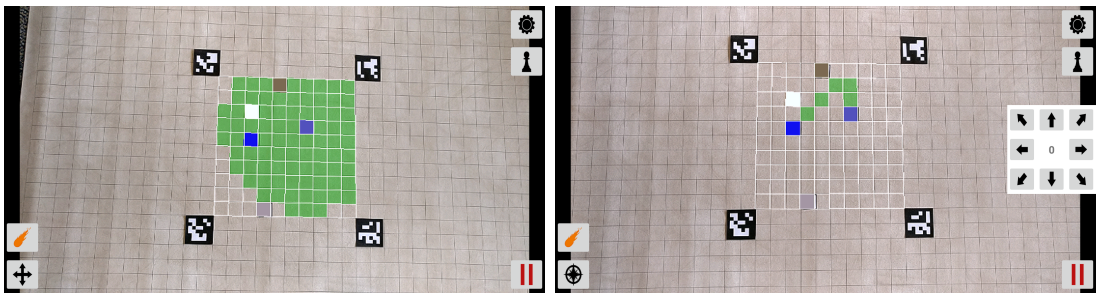

Figure 5.7: The rogue's base movement range and the plotted movement

brings the knight within range of his switchblade and allows the rogue to make an attack.

With the rogue's turn concluded, it's time for the magician (grey square) to act. Being a combat mage, he has several damaging spells prepared and will cast one of the most well known spells in D&D history: fireball. Fireball is an explosion that radiates from a grid intersection and has a blast radius of 20ft (four squares). This should be more than enough to hit everyone in the upper half of the battlefield, but we don't want to do that. One of the characters currently fighting is on our side, so we want to exclude the knight from the blast radius. To find out where we want to position the explosion for maximum effectiveness, we enable and switch the blast template mode to 'sphere' and then set the radius to 20ft. We then experiment by placing the template in several different positions until we find one that fits our criteria of covering the rogues and archer, while leaving the knight excluded.

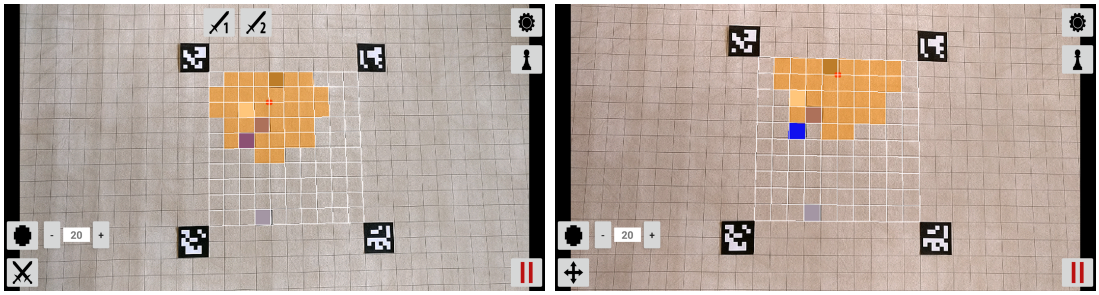

Figure 5.8: The original and adjusted blast template positions

Once the mage's turn is over, it is the last rogue's turn. The rogue marked by the

white square is not having a pleasant time at all, having been hit with both a halberd and a fireball. He is going to step back, out of the knight's range, and attack using one of his daggers. Daggers are able to be thrown, with a range increment of 10ft, so we will select the rogue and display his dagger's attack range.

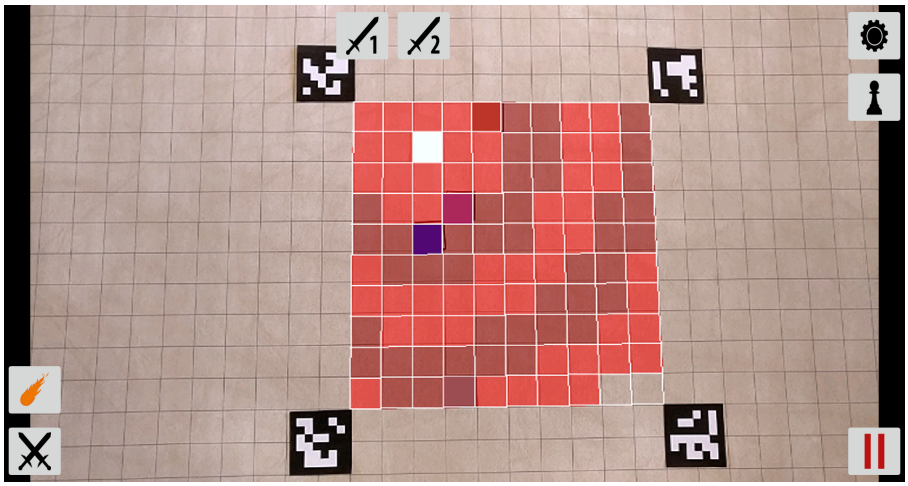

Figure 5.9: The rogue's throwing dagger range

Throwing daggers are usually difficult to hit with, especially against a heavily armoured target, but are also difficult to throw long distances. Ideally, the rogue would like to throw his dagger at the magician, but if he is too far away, he would attack the knight instead. By examining the range increments (shown by the alternating shades of red) in Figure 5.9, we see that the knight is within the second range increment, while the magician is within the fourth. Using this information, the rogue will attack the knight with his throwing dagger since the penalty for attacking a target two range increments away is much less than the penalty for four.

Lastly, the archer (brown square) makes his move. He is equipped with a shortbow which has a range of 60ft. Because of this comparatively long range, he is certain that he can attack anyone on the grid but would like to make sure. By selecting him and showing his attack range, we confirm that he can, in fact, attack anyone within his first range increment. Being the only one who can hit the magician currently, he attacks the

mage and ends his turn.

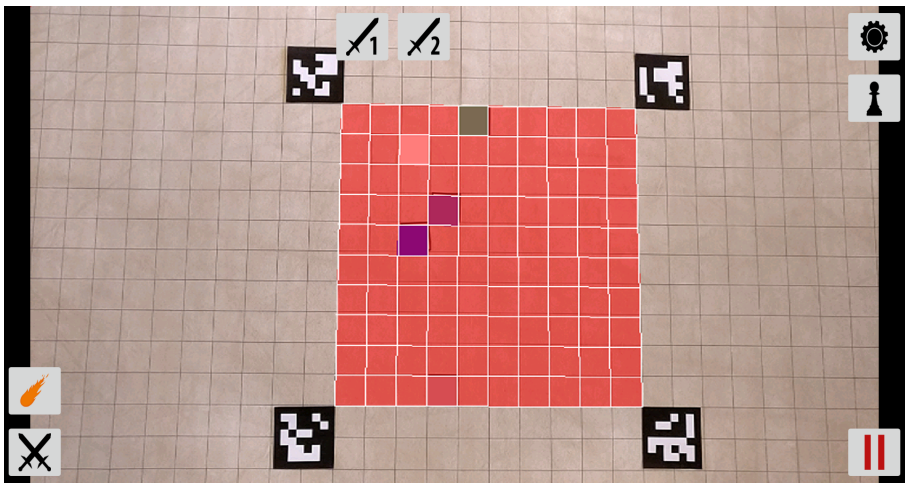

Figure 5.10: The archer's attack range

How the combat resolves itself is irrelevant to this thesis, but this example was used to demonstrate a typical use case for PathfindAR. The app we created was capable of displaying all the information that we wanted and made range finding or movement plotting much easier. During its use we encountered some tracking issues that need to be improved, but the system still remained useful. We believe that the app performed well in this evaluation and has met the goals we set.

Chapter 6

Future Work

There are many different ways that this app could be improved or extended, both from a technical and a feature perspective. Since TTRPG systems are generally complex, there are numerous special cases where the general rules do no apply or are modified. This could be anything from specially shaped blast templates to weapon ranges or movements that vary based on the situation. If we were to extend this tool for the Pathfinder TTRPG system, we would like to add support for these modifiers. Pathfinder is not the only popular system for TTRPGs, however. Currently it is the only system that is supported by the app but the possibility of extending it to other systems, even to ones that use a different style of grid (hex vs. square), is definitely something we would like to consider.

On the technical side of things, there are some issues with the vision algorithms that we would like to improve. Our current implementation assumes that each token occupies only one square. While this is sufficient for most scenarios in Pathfinder, since the majority of monsters/characters fit within a single square, there are situations where the players encounter monsters of different sizes. A creature of size 'Large' in the Pathfinder system, for example, occupies a 2×2 space of squares. Our algorithms do not currently support this, so they would need to be reworked in order to allow for different sizes of

tokens. We would also like to be able to display data from multiple tokens simultaneously. This would be less useful for movement ranges, but being able to see overlapping attack ranges could prove to be useful for planning turns.

For performance, we would like to improve the frame rate that the app can run at. Currently the app can process between 5 and 10fps. While this is usable, it is far from ideal and we would like to improve it. The majority of the app's processing time is dedicated to using the ArUco library to find the grid, so we would like to find (or create) a solution for grid detection that performs better than using ArUco markers.

Lastly, we designed the app for android phones but would like to create an implementation for AR glasses or other wearables. It is safe to say that constantly holding a phone over the game board is not the best solution for the user. If they wish to interact with the game, they need to set the phone down, move their piece, and then possibly re-enter token data if it was lost due to the tracking interruption. AR glasses would solve this usability issue by allowing the user to have their hands free while still providing them with the overlay to show the game information they need.

Chapter 7

Conclusion

Tabletop gaming is a time consuming endeavor, but with the development of PathfindAR, we have created a tool to mitigate some of the more tedious elements of it. Designed for the Pathfinder TTRPG system, PathfindAR tracks tokens, tracks a grid, and displays various ranges for movement, attacks and blast templates. The app is also capable of identifying tokens and storing data tied to them across multiple program executions. Additionally, the app is highly extensible, with a large amount of future work that could be done to support multiple systems, handle special cases in the game rules, and a larger variety of token or grid shapes. Overall, we believe that this app has met the objectives we set at the beginning of this project and has the potential to grow far beyond them.

Bibliography

[1] Android ndk, 2017.

[2] Aruco: a minimal library for augmented reality applications based on opencv, 2017.

[3] Troels L. Andersen, Sune Kristensen, Bjørn W. Nielsen, and Kaj Grønbæk. Designing an augmented reality board game with children: The battleboard 3d experience. In *Proceedings of the 2004 Conference on Interaction Design and Children: Building a Community*, IDC '04, pages 137–138, New York, NY, USA, 2004. ACM.

[4] S. Bedoya-Rodriguez, C. Gomez-Urbano, A. Uribe-Quevedoy, and C. Quintero. Augmented reality rpg card-based game. In *2014 IEEE Games Media Entertainment*, pages 1–4, Oct 2014.

[5] Duy-Nguyen Ta Huynh, Karthik Raveendran, Yan Xu, Kimberly Spreen, and Blair MacIntyre. Art of defense: A collaborative handheld augmented reality board game. In *Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games*, Sandbox '09, pages 135–142, New York, NY, USA, 2009. ACM.

[6] Wonwoo Lee, Woontack Woo, and Jongweon Lee. Tarboard: Tangible augmented reality system for table-top game environment. In *2nd International Workshop on Pervasive Gaming Applications, PerGames*, volume 5, 2005.

[7] OpenCV. About, 2017.

[8] J. Peterson. History forty years of adventure.