GamesPlane: Augmented Reality Gamesman

Miller Hollinger

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2025-136
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2025/EECS-2025-136.html

June 4, 2025

Acknowledgement

To Dan Garcia, for being an excellent research advisor.
To Eric Paulos for kindly reviewing this paper.
To Josh Zhang for his assistance in camera calibration which greatly improved the system's
accuracy.
To Alvaro Estrella for extending the GamesmanUni API.
To Sriya Kantipudi for adding new games to GamesPlane.
To Abraham Hsu for consulting with me about vector math.
To GamesCrafters as a whole.
To Barak Stout, my 9th-grade computer science teacher whose patience and knowledge led me into the wonderful world of computation.
Above all, to my parents, John and Karen Hollinger, for supporting me throughout my academic career and being an endless source of kindness and advice.

GamesPlane: Augmented Reality Gamesman

by Miller Hollinger

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Teaching Professor Dan Garcia
Research Advisor

Date

* * * * * * *

Professor Eric Paulos
Second Reader

Date

Abstract

In this report, I introduce **GamesPlane**, a system that uses augmented reality to overlay the pre-calculated colored value of moves for a player playing a physical two-person, complete-information board game. The values come from application programming interface (API) calls to our Gamesman system that has strongly solved the game. It provides 20 frames per second updates for boards of all shapes and sizes with near-100% accuracy. Finally, it is highly configurable, with tools and documentation provided for easy extensibility.

Acknowledgements

To Dan Garcia, for being an excellent research advisor. His guidance, encouragement, and positivity made GamesPlane possible, and my love for computer science research is thanks to him.

To Eric Paulos for kindly reviewing this paper and his feedback on the project's development.

To Josh Zhang for working alongside me to develop and test the GamesPlane project, and especially for his assistance in camera calibration which greatly improved the system's accuracy.

To Alvaro Estrella for extending the GamesmanUni API to integrate with GamesPlane.

To Sriya Kantipudi for adding new games to GamesPlane and working with me to improve its accuracy.

To Abraham Hsu for consulting with me about vector math.

To GamesCrafters as a whole; for the Monday-Wednesday-Friday lunch meetings and the late-night hangouts. My college experience would not have been the same without them.

To Barak Stout, my 9th-grade computer science teacher whose patience and knowledge led me into the wonderful world of computation.

Above all, to my parents, John and Karen Hollinger, for supporting me throughout my academic career and being an endless source of kindness and advice.

Contents

Chapter 1: Introduction

GamesCrafters is a research group dedicated to solving, analyzing, and playing two-player turn-based complete-information games such as Tic-Tac-Toe, Nine Men's Morris, or Othello. GamesCrafters' system Gamesman [8] accomplishes this goal, using a code description of a game to create a solution. Over time, its scope expanded and it added puzzles and more complex games. A public website, GamesmanUni [9], was created to transition Gamesman away from a download-and-recompile X11-based-GUI model and towards the modern age of web apps. Figure 1.1 shows a view of the site.

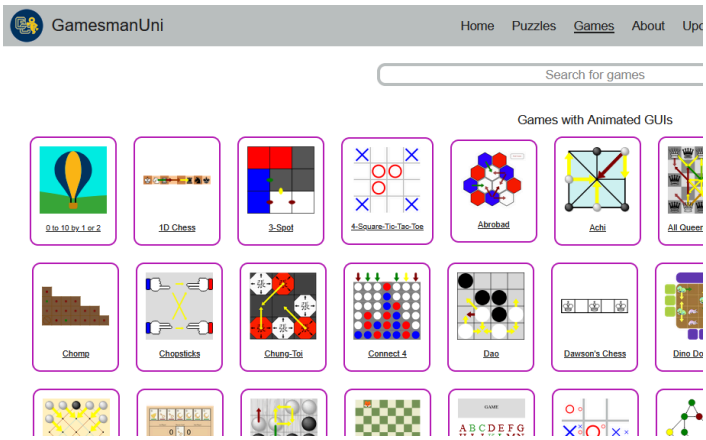

Figure 1.1. GamesmanUni sports nearly 100 playable games and puzzles, all strongly solved.

Despite this growth, Gamesman had yet to cross a critical boundary… into the real world. Although most of the games it solved were nearly always played face-to-face, finding the values of moves through Gamesman required opening a website (or running a local script) to input in each move one at a time, then cross-reference the results with the real-world board.

With GamesPlane, we enter the real world using ArUco tracking-powered boards and pieces. GamesPlane is a system for creating, reading piece data from, and displaying information onto specially designed game boards. Using GamesPlane, you can define the information for a game board, use a camera to locate where the pieces are, and then see the best moves in augmented reality. In doing so, GamesPlane serves as a real-life interface to Gamesman, bringing the perfect play hints once relegated to computer screens to physical game boards.

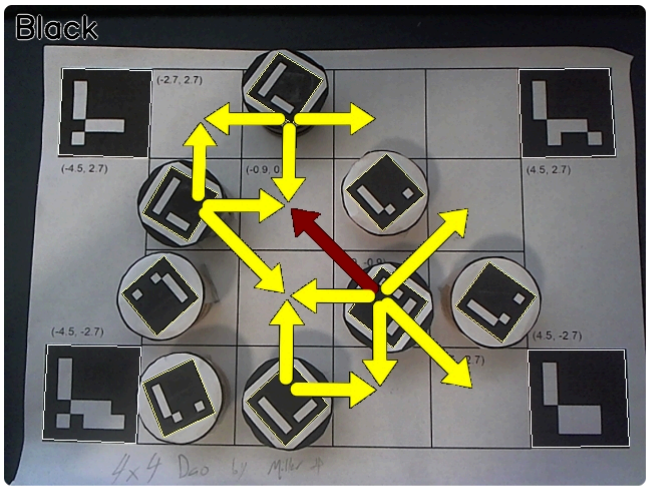

Figure 1.2. The overlay displayed by GamesPlane on Black's turn. Yellow moves are "draw" moves, and red moves are "losing" moves. Green moves are "winning" moves, though no such moves exist in this particular position.

A user of GamesPlane can print out one of many game board and piece sets, then run GamesPlane on their device. GamesPlane finds the pieces and sends that information off to GamesmanUni, which returns an image showing valid moves in the position, with green "winning", yellow "tie" and "draw" moves, and red "losing" moves. This image is displayed in augmented reality (i.e., projected over the camera picture) using information about the board, creating the final result: a real-life image overlaid with an AR graphic showing moves based on the pieces' positions (Figure 1.2).

GamesPlane works using ArUco markers, which are square black-and-white tags featuring specific patterns easily recognizable by cameras. Detection of ArUco markers is implemented by OpenCV, making them an easy tool for computer vision projects such as GamesPlane. A user places anchor markers onto a game board and attaches them to pieces. These markers' relative positions in an image are used to determine where the pieces are in space, and then on the game board.

GamesPlane has the unique quality of being extensible by design: with its integrated documentation, starter guide, and Craftsman tool, future users will have an easy time learning to use the system and set up boards.

The system runs at a high frame rate — at least 20 frames per second, even on low-end systems. There are currently 5 games that function on GamesPlane. Templates are provided to print 8.5x11" boards for a variety of games, but hypothetically any size or shape can function as long as camera quality is high enough. It can fail when ArUco markers are obscured or the camera is at too low of an angle. but it is also fairly tolerant: it functions even when pieces are imperfectly aligned, unevenly rotated, or of different sizes.

Chapter 2: Background

2.1: GamesCrafters, Gamesman, GamesmanUni, and GamesPlane

GamesCrafters' central goal has always been to strongly solve deterministic two-player turn-based games with complete information like Tic-Tac-Toe, or Connect Four. We use a system called Gamesman [8] to solve these games; it exhaustively searches all possible board states for a game, eventually creating a database storing every position and if it is a win, lose, tie, or draw in perfect play (and how many moves it is from the end of the game). Users most often access these databases using GamesmanUni [9], a publicly available website that lets users play board games online through graphical user interfaces (GUIs). During their games, players are shown *value moves*: the moves they can make colored red, yellow, or green for if they are losing, drawing/tieing, or winning respectively. This flow of information is shown in Figure 2.1.

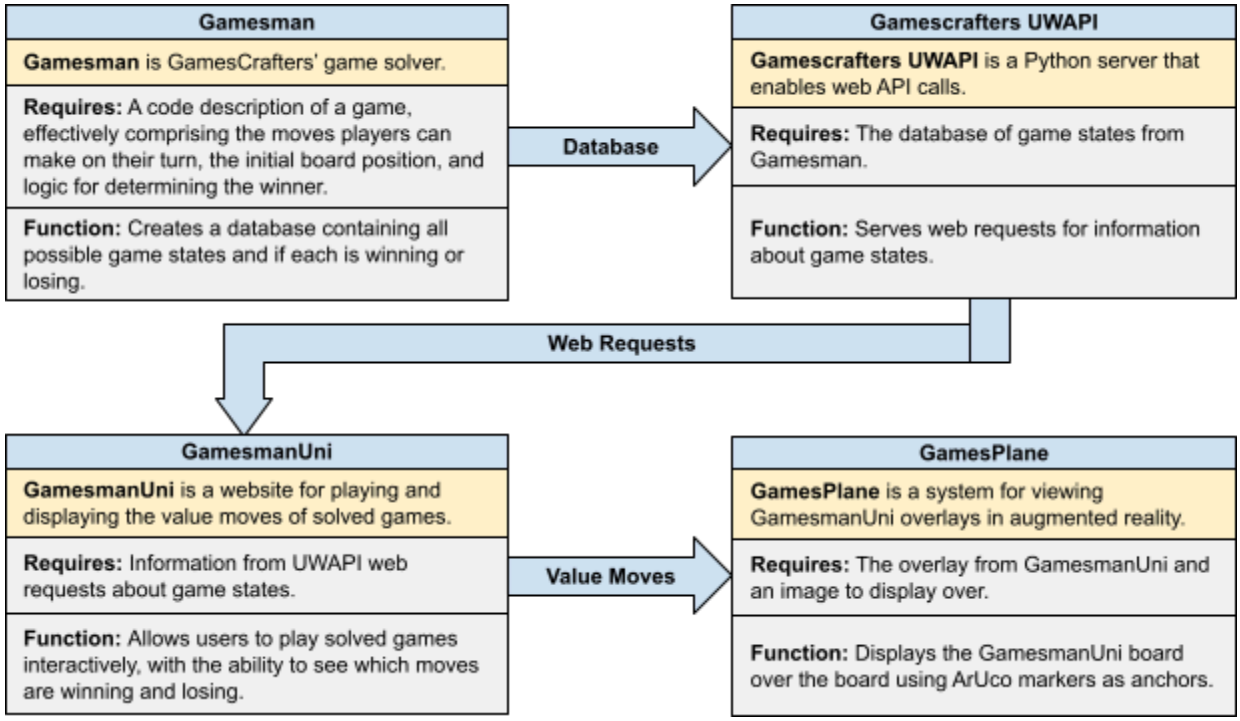

Figure 2.1. Information about the game is generated by Gamesman and then served by UWAPI to GamesmanUni. The user interface created by GamesmanUni is sent to GamesPlane.

2.2: Bringing Gamesman to Physical Board Games

For a long time now, it has been a goal of the lab to bring the power of Gamesman to the physical domain. The reason for this is fairly straightforward: board games are generally played face-to-face, not online. Before GamesPlane, to use Gamesman with a real-life game would require the user to enter into GamesmanUni every move they make, which is a cumbersome approach that slows down the pace of gameplay.

Attempts at crossing the barrier between digital and physical have been made in a few instances. For example, in Fall 2024, a project to display value moves for Connect Four was attempted using color detection. Although this method was successful, it was not generalizable to other games, as it worked by detecting the average color (red or yellow) in each Connect Four board space. Additionally, this system only worked on video recordings, not on live video.

The objective of creating a system that would allow many games to be played in real life remained.

Aside from the goal of crossing into the physical domain and generality to many games, a third goal was accessibility. Another key idea of GamesCrafters is that games should be enjoyed by everyone. GamesPlane, then, aims to be as easy to use as possible for new users. This suggests the inclusion of one-click interfaces, easy setup, and exhaustive documentation. Together, usability features such as these contribute to the ongoing use and development of the software, both by casual and technical users.

Chapter 3: Related Work

The concept of a system that converts video of board games into digital board states is in itself not novel. Multiple papers have been authored on the topic, generally focusing on implementations for popular board games like Chess or Go. GamesPlane's implementation proves itself unique through its ability to be generalized to many board games, rapid response time, and *value move* displays.

3.1: Making real games virtual: Tracking board game pieces [1]

This student project from USCS makes use of RANSAC and Hidden Markov Models to find the grid of a Go board, and then determine the colors of pieces on the junctions between spaces. The system is functional on standard unmarked Go stones, even at off-angles, and is able to detect the entire 19x19 Go board. It has good detection accuracy at about 91%. Its accuracy is boosted using an A* algorithm that uses previously known board states to predict the likelihood of detected board states: if a state is detected but is many moves away from the initial state, it is deemed less likely and a simpler explanation is used. The main drawback of this system is its time to operate, taking 40 seconds to run 20 iterations of an A* algorithm per picture. It also does not have any real-time board overlay, instead only focusing on recording the game over time.

3.2: Augmented Reality Chess Analyzer (ARChessAnalyzer) [2]

ARChessAnalyzer makes use of a Convolutional Neural Network (CNN) approach alongside an AR overlay to show recommended moves overlaid on images of chessboards. Their CNN approach enables them to use existing, unmodified chess boards and pieces. They tout a 93.45% accuracy in state recognition, which is excellent considering that up to 32 pieces must be recognized and positioned correctly for a board state to be correct. When compared with some of the other related works, this accuracy is especially impressive. However, the complex calculations involved in running a CNN results in a 3-4.5 second waiting period between taking a picture of the game and seeing an AR overlay. Considering that augmented reality relies on realistically superimposing the digital world over the real world, this delay becomes a notable drawback as it breaks the appearance of the digital objects appearing "in real life."

3.3: An Intelligent Chess Piece Detection Tool [3]

This paper uses CNNs to locate and categorize chess pieces, with the goal of creating a board state. The paper focuses primarily on the method it uses to categorize pieces: a YOLO object detection algorithm. YOLO means "You Only Look Once," and refers to an algorithm that runs the image through its network a single time. Before YOLO, approaches such as R-CNN were used which would often need to propagate a single image through a network thousands of times. With YOLO, faster or real-time object detection becomes possible. Found chess pieces are displayed on a separate chessboard to the side of the image of the game board. It has acceptable accuracy per-piece at 84.29 percent correct in the best case, however, full board state detection can be erroneous considering up to 32 pieces can be on a chessboard at once. It requires the use of a custom CNN, which takes between 2 and 12 hours to train and 3 to 21 seconds to run. Considering that it also requires a massive dataset (about 140,000 pictures in the case of chess) to train, it becomes very inefficient to convert this CNN-based approach to other games.

3.4: Chessboard and chess piece recognition with the support of neural networks [4]

This paper from the Institute of Computing Science at Poznan University of Technology uses a novel lattice detector to find a chessboard in an image, and then a support vector machine and a convolutional neural network to locate the pieces. They make use of a chess engine to determine what board positions are most likely to increase accuracy (e.g. having three white bishops is very uncommon). With this approach, they achieve an astounding 99.57% accuracy in chess board detection and 95% accuracy in piece detection. Drawbacks of their approach include the fact that it does not transfer well to other games, require a lattice-shaped board, and its execution time, which sometimes reaches 4.5 seconds for a single frame.

3.5: Chess Piece Detection [5]

This approach uses a YOLO CNN to detect the spaces of a chessboard as well as the pieces on it. Generally, its approach is split into three steps: a “board detection” step where the edges of the board are located, a “grid detection” step where the spaces of the board are delineated, and a “chess piece detection” step where individual pieces in each space are recognized. It leverages the fact that chessboards have alternating light and dark colors to detect a variety of boards using OpenCV. As with the other CNN approaches, the major drawback in this case is that it relies on a specifically trained CNN designed for chess pieces, and so conversion to other games is a costly affair. Additionally, its use of contour tracing means the approach can only practically function on chessboards or other square boards/

3.6: Comparison to GamesPlane

Considering previous work in the field of game board recognition and move display, GamesPlane differentiates itself in a few critical ways.

Generality: GamesPlane can function with practically any game, not just Chess and/or Go. Even games that are not compatible with Gamesman can still have their board states extracted. Critically, this applies even for games with irregular boards: any board shape (e.g. hexagonal, triangular, or completely irregular) can function with GamesPlane. Pieces with flat tops (like checkers, disks, or tiles) work best with GamesPlane, but other shapes can work as well as long as ArUcos are flat when attached.

Accuracy: Thanks to its ArUco-based approach, GamesPlane has excellent accuracy at a variety of angles, while other systems tend to focus on functioning at one specific angle. With the camera positioned from the top down, its recognition accuracy is essentially perfect. Accuracy is extremely important in the case of strategy games, as even a single misplaced piece can completely alter what the best move is.

Speed: GamesPlane runs at a high frame rate, around 25 frames per second. Additionally, there is no visible delay between real life and the video display, even on the low-end laptop used for testing the project. A higher-end computer could hypothetically achieve even higher frames per second. It only has to query GamesmanUni once to display AR value moves, however, so this is still generally faster than the CNN-based approaches. Additionally, GamesPlane can cache its overlays, leading to instant response times.

AR Overlay: By using GamesmanUni’s overlays, GamesPlane gains access to the interface of every game implemented in GamesmanUni without needing to write additional interface code. That is, we don’t have to re-draw the interface’s arrows for sliding moves and dots for placement moves. We instead use the graphic drawn by the existing system and overlay that, leveraging abstraction and a “don’t repeat yourself” software philosophy. Additionally, because Gamesman solves games fully, our overlay shows perfect value moves, not algorithmic or AI suggestions.

A downside of our system is that the user must prepare a GamesPlane board with ArUco markers. However, with a pre-made PDF ready to print, even this process takes no more than a few minutes per game. Existing game boards can also be converted to GamesPlane compatibility by adding ArUco anchor markers. GamesPlane therefore makes a worthwhile tradeoff that results in a variety of benefits valuable to board game players.

3.7: System Comparison Chart

System	Applicable Games	State Recognition Accuracy	Operation Speed	Value Moves Display	Underlying Technology
Making real games virtual: Tracking board game pieces	Go	90.57%	40 seconds per frame	None	RANSAC, Markov Models, A*
Augmented Reality Chess Analyzer (ARChessAnalyzer)	Chess	93.45%	4.5 seconds per frame	Displays chess engine moves	Convolutional Neural Network
An Intelligent Chess Piece Detection Tool	Chess	84.29% per piece	3-4 seconds per frame	Separate board display	Convolutional Neural Network
Chessboard and chess piece recognition with the support of neural networks	Chess	95%	4.57 seconds per frame	No display; only locates pieces.	Support Vector Machine, Convolutional Neural Network
Chess piece detection	Chess	81%	Real-Time	No display; only locates pieces.	Convolutional Neural Network
GamesPlane	Any Gamesman Game	97-100%	Real-Time	Displays value moves in augmented reality	ArUco with OpenCV, Gamesman

Chapter 4: Development and Challenges

4.1: Conceptualization of GamesPlane

When creating the initial concept for this project, I knew I would need a board that would provide some kind of visual anchor that a camera could detect. I'd seen AprilTags [11] from robotics applications before, and so decided to use ArUco tags as they're integrated into OpenCV [12]. Python is a language already widely used for various applications in GamesCrafters, so using Python with OpenCV was a natural approach as it would ensure future GamesCrafters members would be able to read the code.

4.2: First Iteration

GamesPlane itself began as a single object, called "The GamesPlane." This was a wooden board, about a foot to a side, featuring a 5-by-5 grid with 2-inch wide squares (see Figure 4.1). It is compatible with games such as Tic-Tac-Toe, 4x4 Othello, Chung-Toi, or about 10 others supported by Gamesman. The board had special slots in which 5 ArUco markers could be placed. I assumed that the ArUco tracking would be very erroneous, and that having more markers would essentially allow for multiple samples.

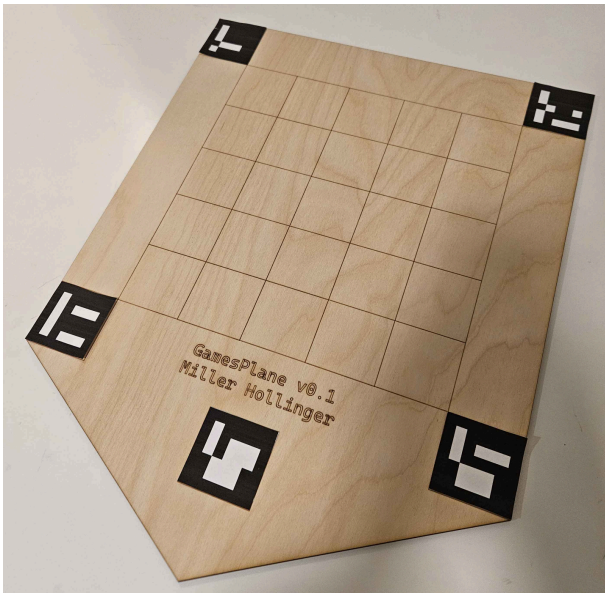

Figure 4.1. The original GamesPlane: A wooden board, intended to be the only board compatible with the system.

For the pieces, I 3D-printed custom pieces. As seen in Figure 4.2, these were styled in the shape of a rook, but shorter. My thought process was that having tall pieces would obscure the ArUco markers pasted onto the board when viewed from an angle, so shorter pieces were preferable. At the time, I was unsure if I would be placing ArUco markers onto the pieces or if I would be using some kind of image detection, and so I kept the tops flat to allow space for an ArUco marker to be attached. I additionally added ridges onto the edge in the hope that it would make the piece's shape more defined, making it easier to detect with any image detection method I might use.

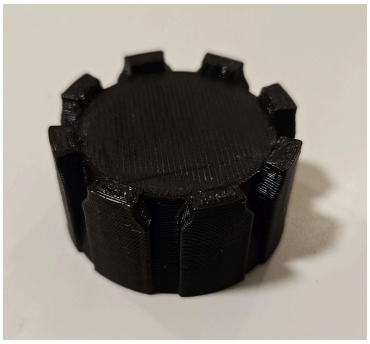

Figure 4.2 The original GamesPlane piece, meant for use with Haar Cascades [6] as explained below. It was meant to evoke the shape of a chess rook.

The first pipeline I envisioned for the GamesPlane was as follows: the ArUco markers would have hardcoded real-life positions. I would then use an object detection method called Haar Cascades [6] to locate the pieces in the frame, and use their positions in the frame compared to that of the ArUcos to estimate their real-world position. The reason for this decision was that I wanted the game pieces to look like normal game pieces — I thought that adding ArUco markers on top of them would make them look too different from traditional board game pieces.

4.3: Transition to ArUco-Only Approach

Unfortunately, this initial approach worked very poorly. The main issue was with the Haar Cascades. In order to detect a new object, it's necessary to train a new cascade using images containing the object (positive examples) and images not containing the object (negative examples). I therefore took about 250 pictures of the piece around my apartment, and used images from the internet as negative examples. This failed to adequately train the cascade: it ended up with a false-positive rate of about 70% on images not containing the piece. The issue was down to the variety of images I fed in. By taking many similar pictures, I inadvertently trained the cascade to detect a few much simpler features: specifically, a shadow that appeared on the edge of my table and another shadow that appeared alongside the wall (see Figure 4.3). With how many situations create shadows similar to these, it would regularly find the piece in locations it was not. At this point, the two ways forward were either to take much more varied pictures of the piece in many contexts or to simply change to an all-ArUco approach. Considering the scope of the project, it seemed more sensible to switch to relying on ArUcos entirely.

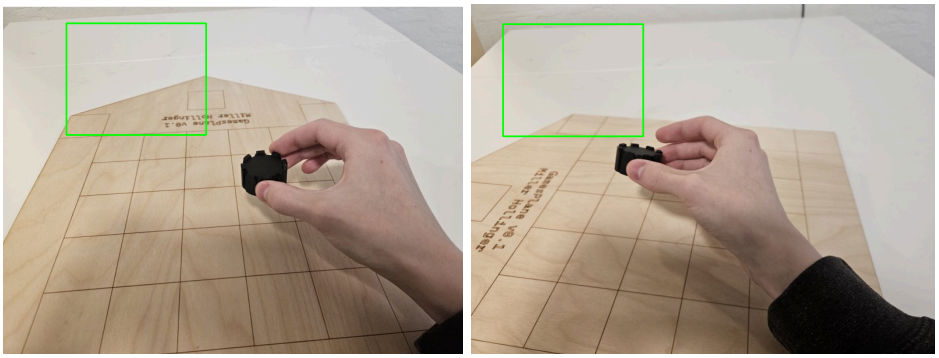

Figure 4.3. The Haar-based approach had very low accuracy, and would usually detect a random crack in the table. The green square is where the Haar classifier thinks the piece is — it's actually in my hand.

I began by attaching ArUco markers to the top of the 3D printed pieces. However, there was an issue here as well — ArUco markers need white space around their edges. When detecting a marker, contrast between white and black is used, and so if the marker does not have a white border it fails to detect it. The tops of the pieces were too small, so placing a marker with a border on top of them made the markers invisible to the camera from even a short distance. To rectify this, I stopped using the pieces and started using the ArUco markers on their own with no piece supporting them — this way, at least while testing, they would be larger and more visible.

At this point, with a solely ArUco-based approach, I went about implementing OpenCV's ArUco detection. The challenge it presented was one of coordinate spaces. My final goal was to extract what I call "board coordinates": where a game piece is on a board (e.g. f4 in chess). To get these coordinates, I'd need to first obtain board-centered world coordinates, which are (x, y, z) coordinates for where a game piece is in space aligned to the game board's frame of reference. These would then come from camera-centered world coordinates, which can be obtained by estimating the relative position and pose of an ArUco marker to a camera by using an image the camera has taken. Converting between these four coordinate spaces, and displaying information from each in a sensible way, made up much of the work of the project. Small mathematical errors were difficult to detect but had a massive effect on the accuracy of piece location.

After resolving these conversions, I finally was able to locate a piece, but inaccurately. The detection would regularly place a piece several spaces away from its true location. In strategy games where a single space can make a massive difference in the best move, this was expectedly unacceptable. The breakthrough in raising accuracy came when discussing the camera's calibration file. A calibration file contains the information about a camera's intrinsic properties, specifically its output image size and focal length. Analyzing the file revealed that the camera had been calibrated very poorly, possibly due to an issue in the GitHub repository I used to calibrate the camera or my use of it. It estimated the camera's field of view as being ten times its actual size. I then replaced the calibration matrix with a matrix found on the internet for a similar camera, and immediately the piece accuracy increased from 15% to 55% correct.

4.4: GamesPlane Architecture

With an acceptable level of accuracy, the next step was to break down the large Python file that ran the pipeline into several modular files.

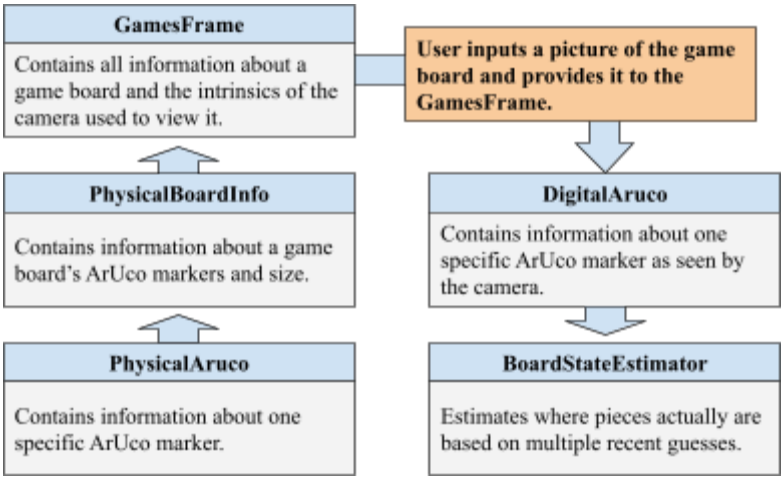

Figure 4.4. The architecture of GamesPlane's board state estimation system.

As shown in Figure 4.4, for each game, a list of `PhysicalAruco` objects are made by the programmer. These objects store information about each ArUco that will be used with the game, such as their size, position, and appearance. Then, these are placed into a `PhysicalBoardInfo` object, which also contains information about the game board itself, such as its size and where pieces may be placed. That object is added to a `GamesFrame` that takes in information about the camera used to play the game. The `GamesFrame` serves as the entry point for queries about the game. When a picture is provided, the ArUco markers in the image are recognized and information about them is stored in `DigitalAruco` objects. This includes where the markers are in the original image, and where the markers are in 3D space. By comparing the relative locations of unmoving anchor ArUcos to mobile piece ArUcos, the locations of pieces on the board — and therefore the board state of the game — can be found. This board state is treated as a guess, and given to a `BoardStateEstimator` object. This object compares all recent guesses to make a more informed guess, using data from a short time window.

4.5: Printable GamesPlanes

Now that the system was modular, the next natural step was to attempt to apply the system to different games and board shapes. At first, I continued to laser-cut wooden boards and tape ArUco markers onto them. Originally, my intention was that all GamesPlanes would be wooden boards. However, this process failed the goal of accessibility — the vast majority of students in GamesCrafters have never used a laser cutter before, and so only creating template files for laser cutting would hamper the development of new boards. For this reason, I switched to using a simple printed-paper based approach with black-and-white PDFs as shown in Figure 4.5. Unlike laser cutters, printers are ubiquitously available. Additionally, it becomes possible to print the ArUco markers directly onto the board. Laser-cut wooden boards were unable to engrave marks dark enough to be detected by OpenCV's ArUco detection, while printed boards had sufficient contrast.

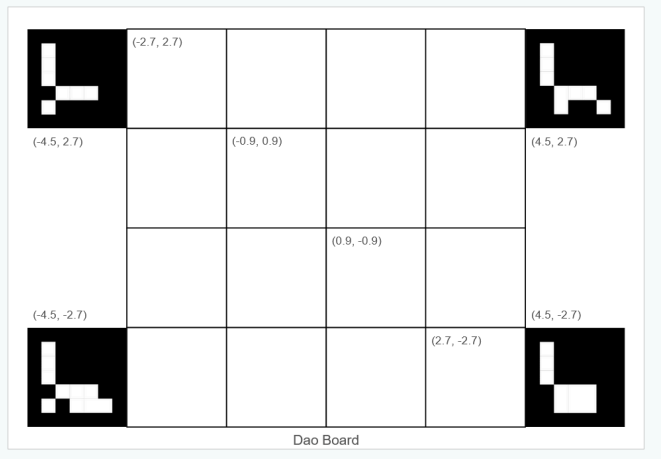

Figure 4.5. The printable Dao board used as the "gold standard" board for this project.

Alongside this printer-based approach came the creation of new template files. Students in GamesCrafters already regularly use Google Slides to create GamesmanUni SVG assets, and so I created a template for creating a new GamesPlane in Google Slides. The template includes four ArUco markers on the corners, which after testing proved sufficient to achieve good accuracy.

4.6: Software Development

While developing the physical form of the board, I also continually developed the software. With the architecture built and functional, my objective was to make it easy to work with. For this reason, I chose to develop GamesPlane as a Python web app running off Streamlit [13]. Streamlit is an easy-to-use Python library that enables quick creation of web applications. I split my application broadly into three parts. The first part is the viewer, which lets users run GamesPlane's live video feed. The second is **Craftsman**, which helps users set up new games with guiding hints. Finally there is documentation, which explains the project and how to get GamesPlane working.

Another approach to raise accuracy was the `BoardStateEstimator` class. This class, designed as an extensible interface, records the system's guess at the board state based on an individual image. It then combines that guess and other recent guesses to make a more accurate estimator. The final approach for this system makes use of a 100-frame sliding window, where the most recent 100 estimates (about 2 seconds of data) are used to create a final estimate that is treated as the truth.

A second approach to improving accuracy was the use of a closer-weighting function. By default, the system works by comparing each anchor ArUco's position to a piece's to get an estimate relative to the anchor, and then taking the average of those estimates. However, running tests with 4-ArUco boards revealed that the closer an ArUco was to a piece on the board, the higher its accuracy was. Therefore, Sriya Kantipudi and I wrote a function that weighted the ArUco markers closer to a piece higher than those further away when taking the average. This raised accuracy at sheer angles where estimates for further-away ArUcos could be incorrect.

These two approaches raised accuracy to about 70%. Although this accuracy was acceptable for testing, it ultimately would not suffice for practical use. The issue, we concluded, was likely still one of calibration. For this reason, GamesCrafters member Josh Zhang created an automatic calibration function for GamesPlane that uses the dimensions of the camera to

make an approximate matrix. After testing, this raised accuracy to about 95%, and practically 100% when viewed from above.

With tracking accuracy improved, the next step was to use the board positions of the pieces to make a web request to GamesmanUni and display the result. The route, written by Alvaro Estrella, takes in a string representing a board game's position and returns an image showing the moves that can be made on the game's board. This image is then displayed over the board using the positions of the board anchors. The result is a board overlaid with value moves as shown at the end of Figure 4.6.

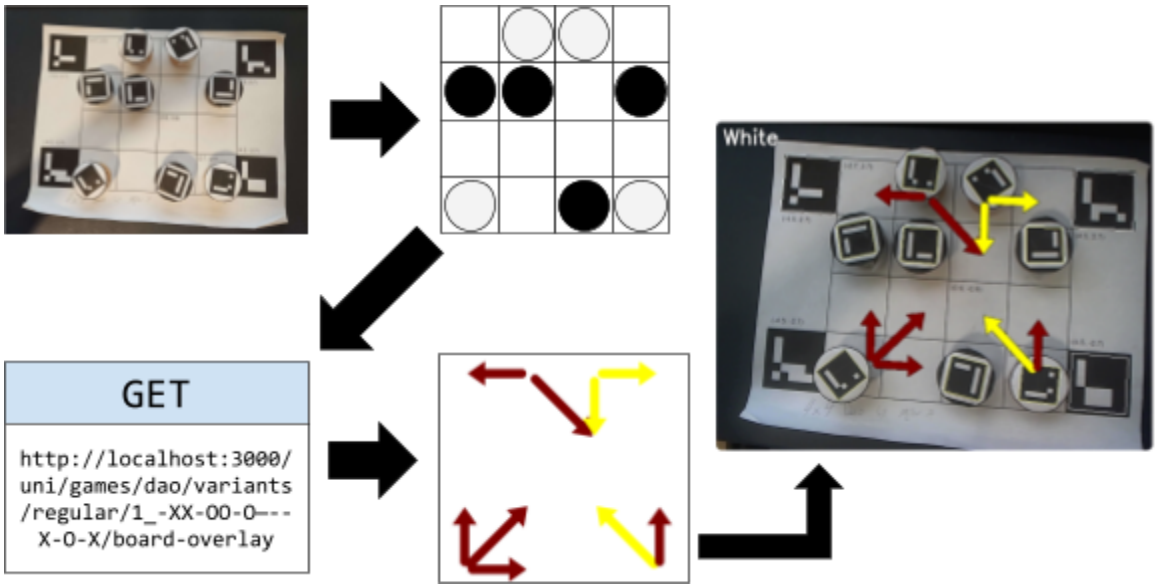

Figure 4.6. An image is converted into a representation of its board. The board is used to make a UWAPI board representation string that is added to a web request sent to GamesmanUni. The returned overlay is projected over the board in AR using the positions of the anchor ArUco markers in the frame.

Python dictionary caching was used in various circumstances on the website to improve performance. First, web requests are cached. This works well because board game states are discrete and determinate. This means that if players play multiple games, it is unnecessary to make the same requests again for earlier board states. It also works well with "loopy" games where players may end up in the same board state several times for a single game. These cached board states are saved to a "pickle" file that is loaded the next time the game is played to retain the cache between sessions. Finally, the locations of the board's ArUco anchors are recorded, which allows the board overlay to be displayed even when all four ArUcos are not visible.

With these improvements, the final step was to clean up the application and write documentation. Documentation includes entries on each of GamesPlane's classes, setting up the system, creating new boards, and troubleshooting. This includes a separate new user guide with instructions to get the system running. The system and all its documentation are now available on GitHub [10] for GamesCrafters members as well as members of the general public.

4.7: Home Page

GamesPlane is a locally run web-app, packed with features and information to help educate users.

The home page shown in Figure 4.7 features the GamesPlane logo and guiding information to other parts of the website. The GamesPlane logo shows GamesCrafters' logo — the "Oxsi" — on a tilted square. Oxsi represents the AR overlay created by GamesPlane, and the square represents the game board the overlay is shown over.

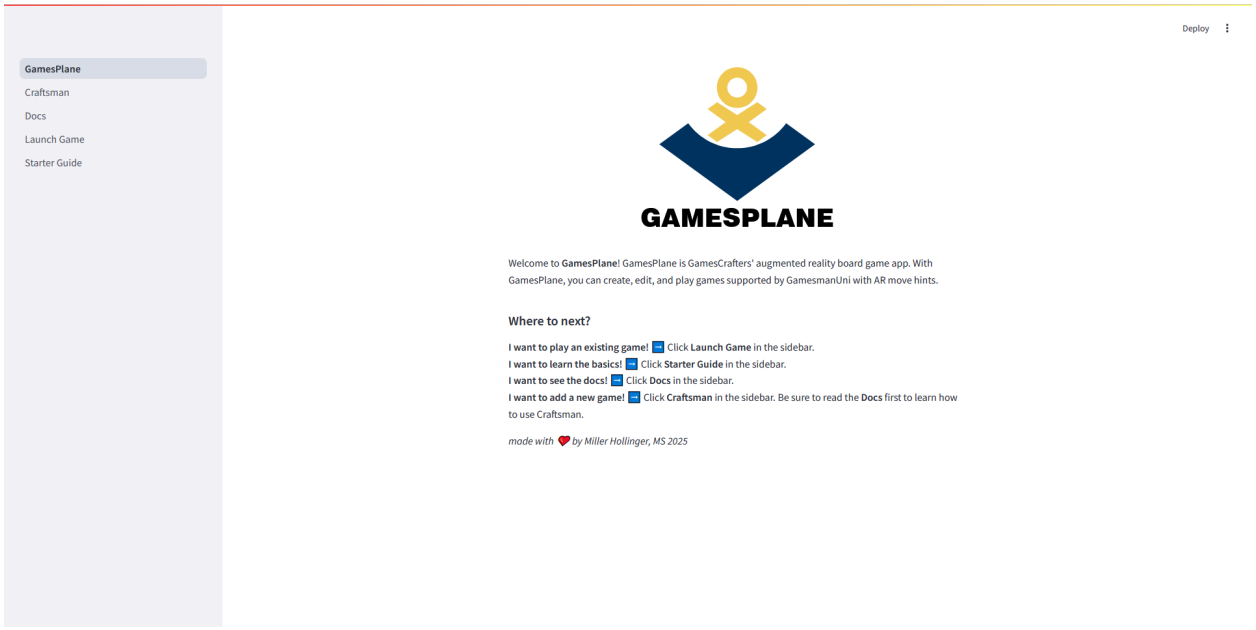

Figure 4.7. The GamesPlane home page.

4.8: Craftsman

Craftsman is GamesPlane's system for creating new games. It allows the user to define everything about the game as shown in Figures 4.8, 4.9, 4.10, and 4.11, including providing a PDF of the game board itself for users to print.

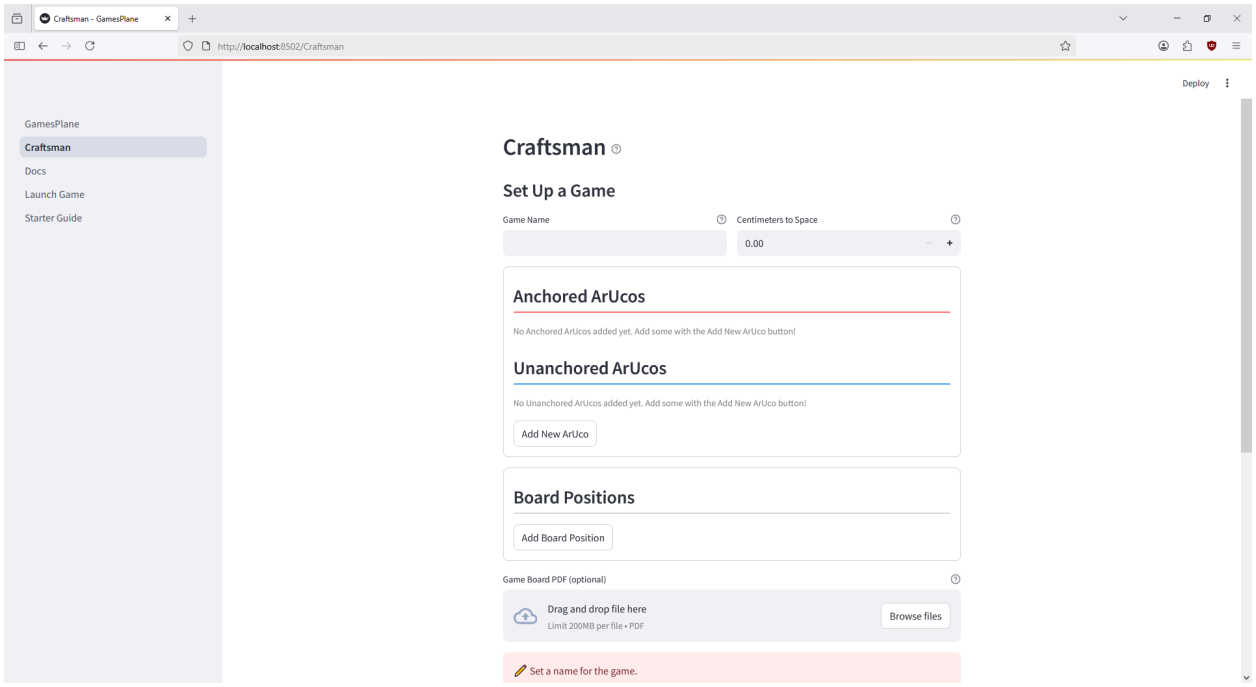

Figure 4.8. Craftsman allows users to set up information about each game.

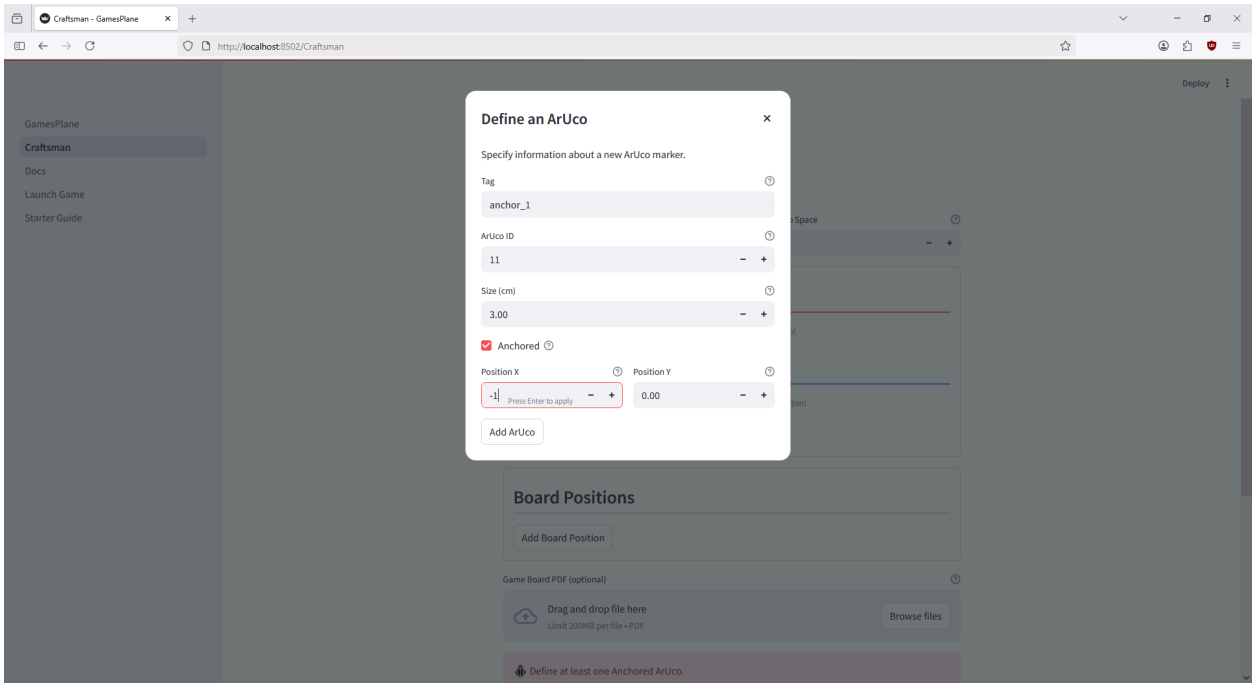

Figure 4.9. Users can define ArUco markers using a pop-up.

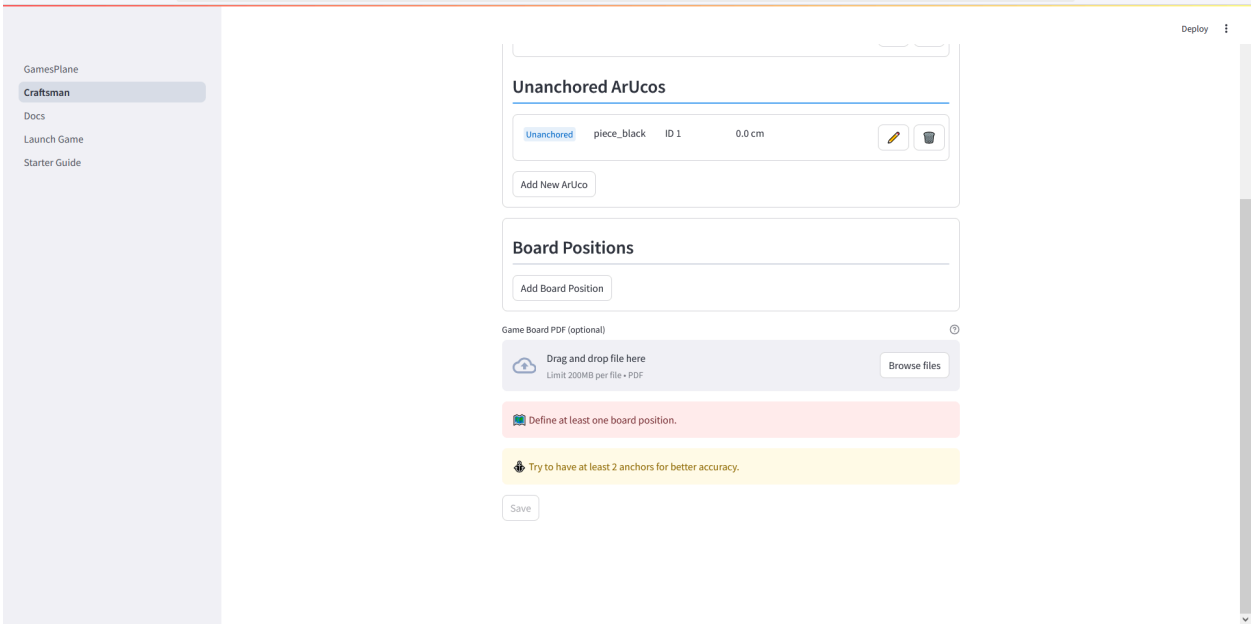

Figure 4.10. Craftsman provides hints to the user to guide them through the process. Red hints are requirements to continue, while yellow hints are suggestions.

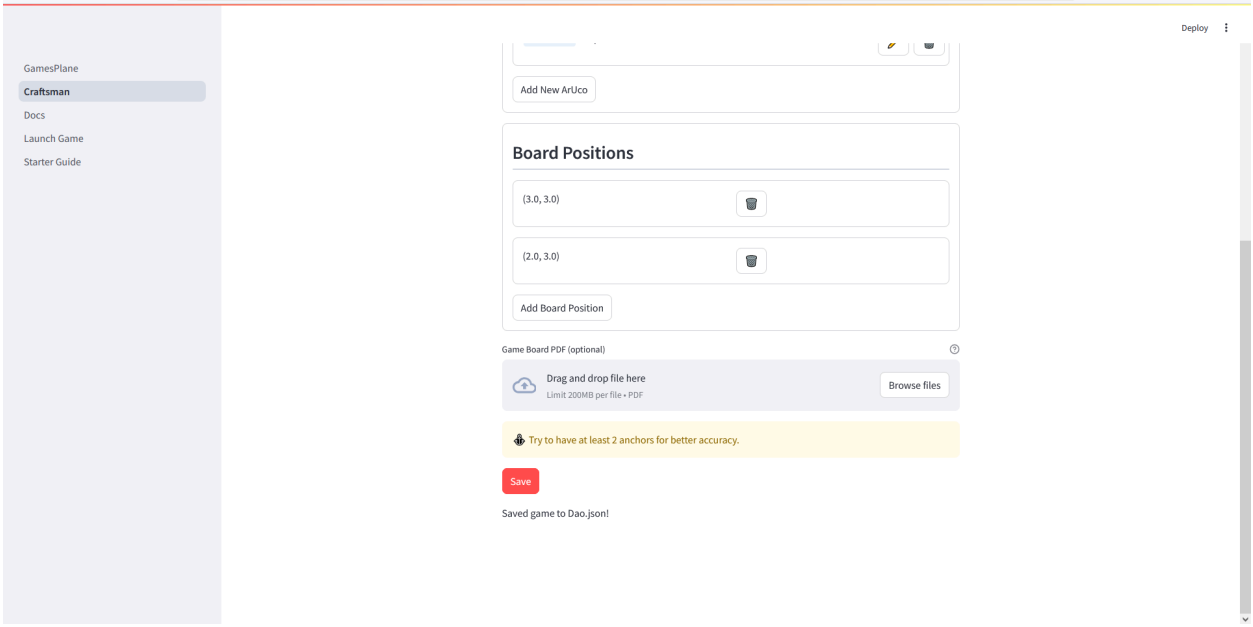

Figure 4.11. When ready, the user can click “Save” to write the data to a project data JSON file. Then, the app can load that information to play the game locally.

4.9: Documentation

GamesPlane has integrated documentation for its various classes (see Figure 4.12), so users who want to add new games or develop new features have a wealth of information to work off of.

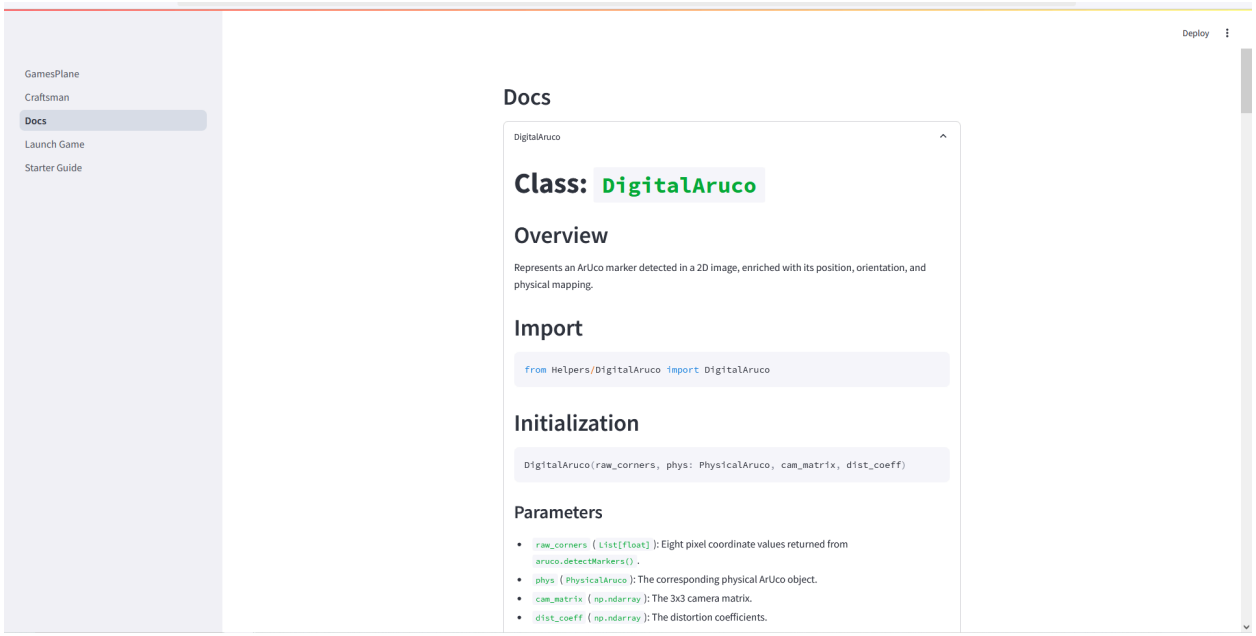

Figure 4.12. The documentation for GamesPlane includes descriptions of each class in its architecture.

4.10: Launch Game

The Launch Game panel is the primary place users engage with GamesPlane. This page lets users see GamesPlane's AR overlays. The user begins by clicking a game to play as shown in Figures 4.13, 4.14, 4.15, and 4.16, and 4.17.

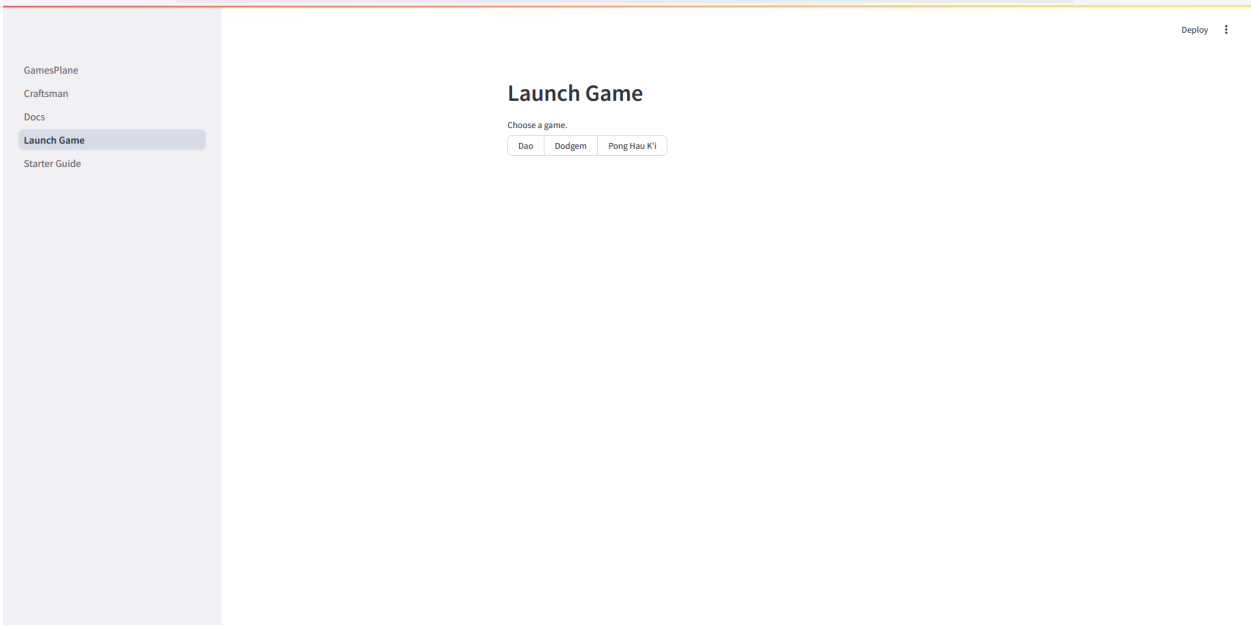

Figure 4.13. The user can click any game set up with GamesPlane to get started.

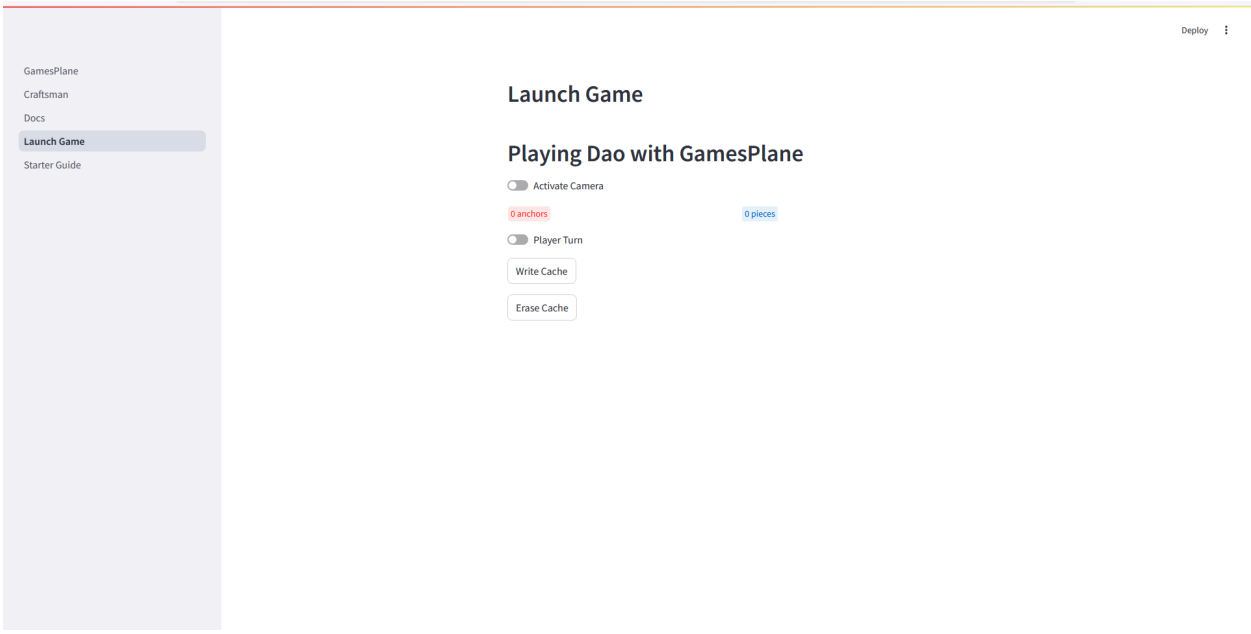

Figure 4.14. Once the game is chosen, the camera is connected and JSON data from Craftsman is loaded. Now, the user can toggle Activate Camera to start the camera feed.

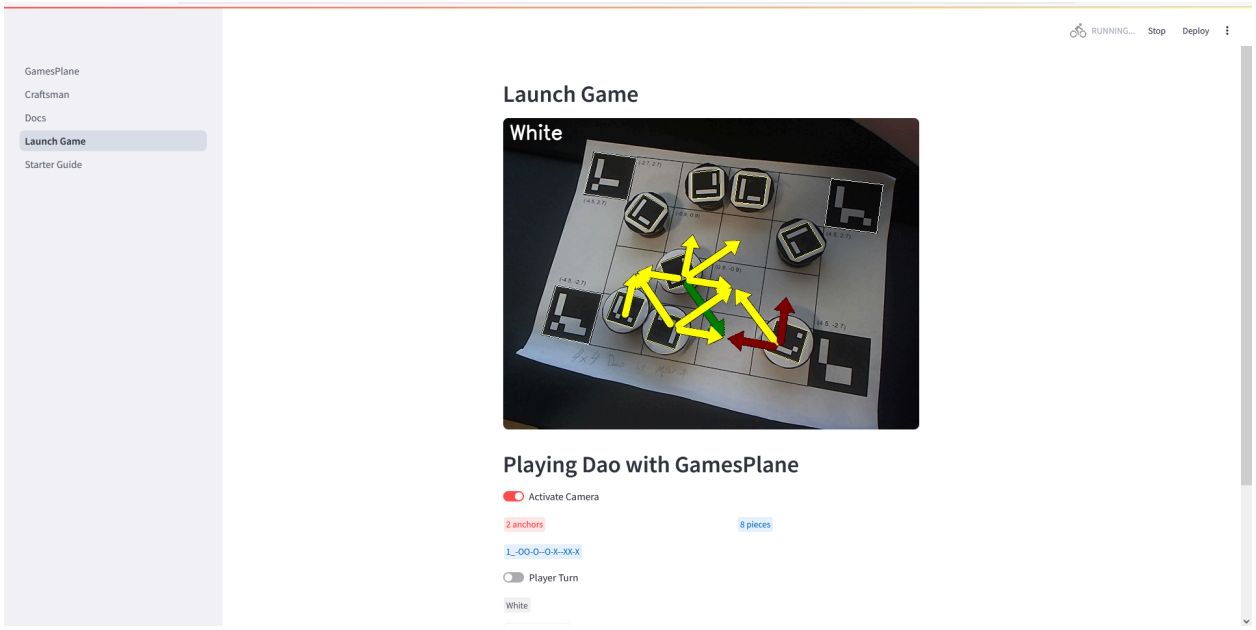

Figure 4.15. The player whose turn it is is shown in the top-left. The AR display is shown over the game board. Information about the number of detected ArUco markers and the extrapolated gamestate are shown as well.

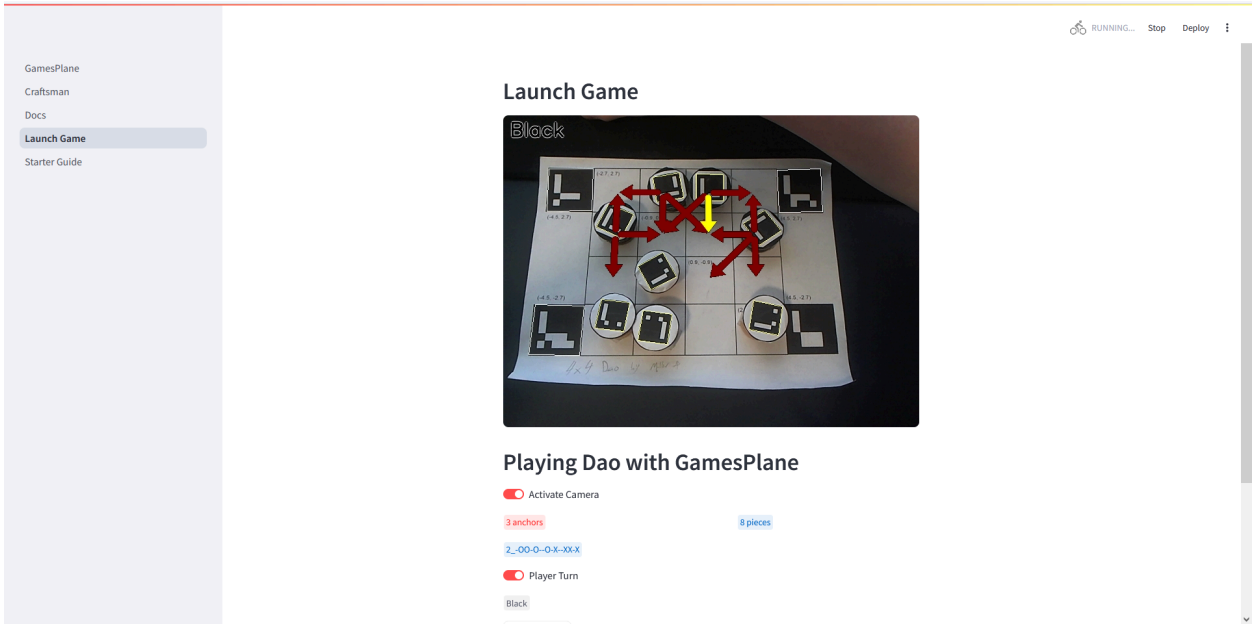

Figure 4.16. The turn can be changed over to switch which pieces have move overlays.

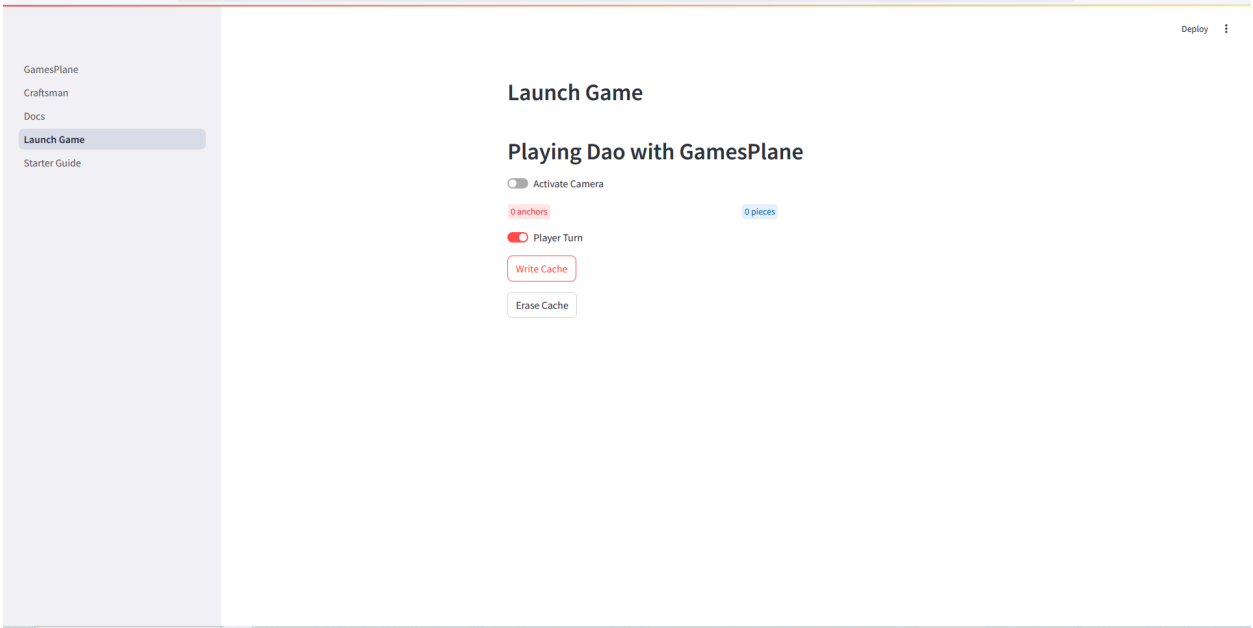

Figure 4.17. The player can also click "Write Cache" to save cached board states to a .pkl file. This file is automatically loaded when the game starts, saving GamesmanUni requests. They can also click "Erase Cache" (if an error occurs) to reset cached information.

4.11: Starter Guide

Lastly, the Starter Guide shown in Figure 4.18 walks new users through how to begin using GamesPlane with an existing game. It also has helpful troubleshooting tips with common issues.

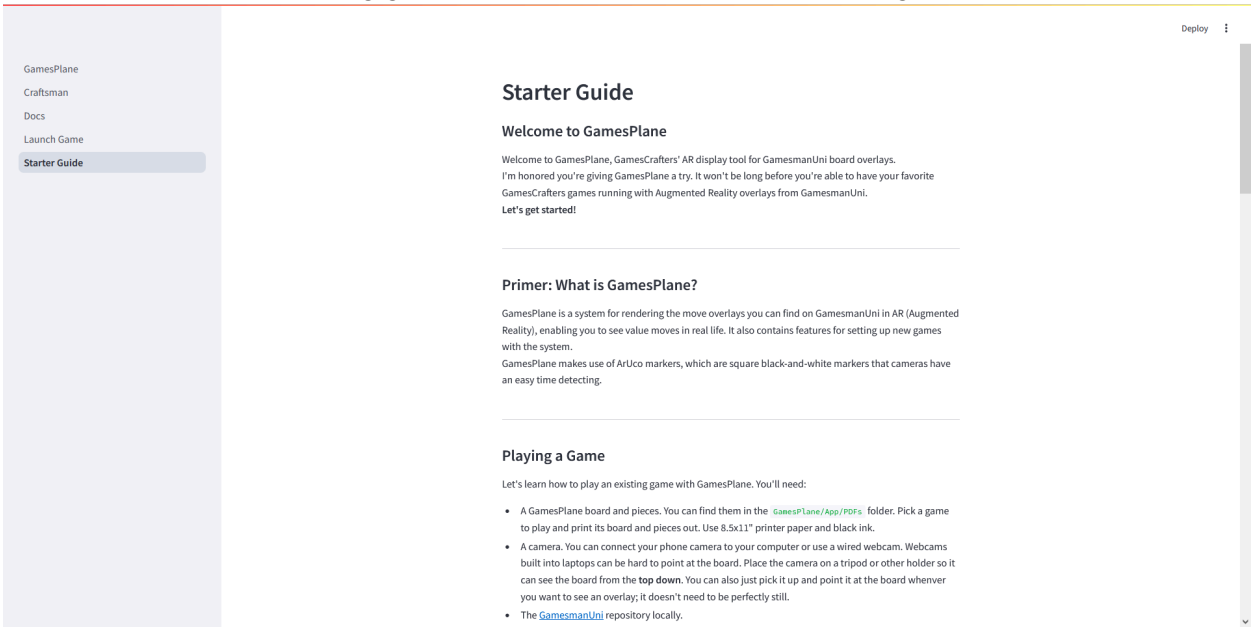

Figure 4.18. The starter guide provides useful information to explain how to play a game.

Chapter 5: Results

5.1: Board State Detection

GamesPlane's primary feature is fast, accurate detection of board states. The following serves as an analysis of its accuracy, including in a variety of edge cases. In sum, it is shown that GamesPlane is accurate even in unusual edge cases. It only fails when the camera is placed at an awkward angle.

5.2: Optimal Accuracy

GamesPlane functions best from a top-down perspective aligned with the board. For my Logitech HD Pro Webcam C920, it fit the entire board into the frame well when placed about 15 inches above the board (for a 8.5x11" printed board). This can be achieved by attaching a webcam to a tripod. Additionally, pieces must be aligned reasonably to the center of the squares they are meant to lie in (see Figure 5.1 below). The pieces may be rotated any way as long as they remain flat on the table (i.e. parallel to the board's anchor ArUcos). Under these conditions, GamesPlane's accuracy is essentially perfect. Out of 100 tests run on Dao positions, all 100 returned the exact correct gamestate. See the below images for examples of these perfect reads.

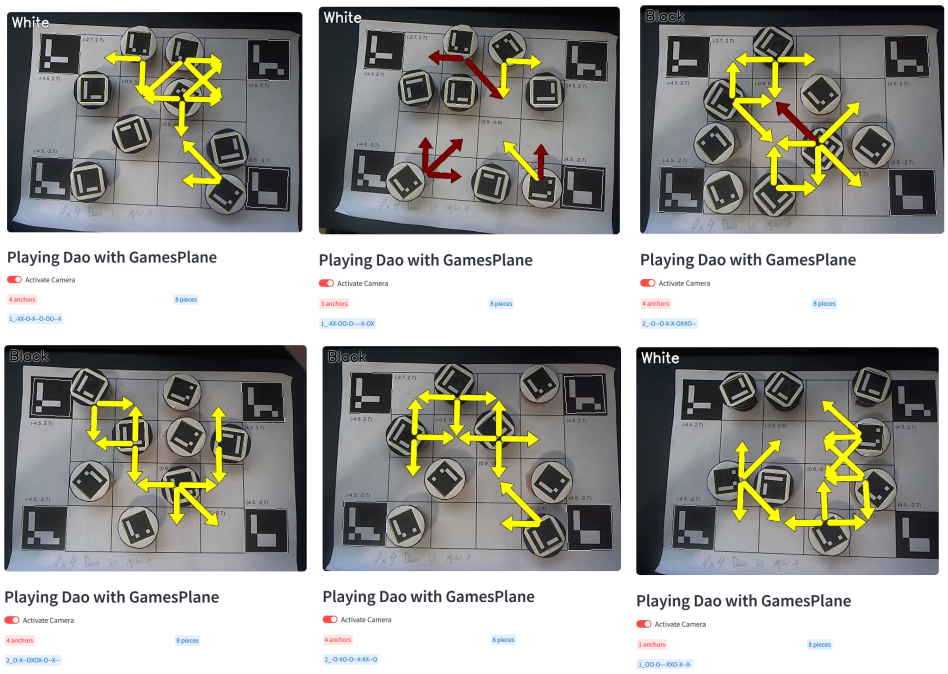

Figure 5.1. These are 6 of the 100 tests run from an optimal or near-optimal setup. Notice that despite slight variance in piece rotation, board rotation, and anchor visibility, accuracy is still perfect.

5.3: Height Tolerance

GamesPlane functions regardless of the height of game pieces, assuming they are all still in the frame. In Figure 5.2, one of the pieces has been placed on a deck of cards to lift it up. Despite this, GamesPlane still recognizes it and extracts the correct board state.

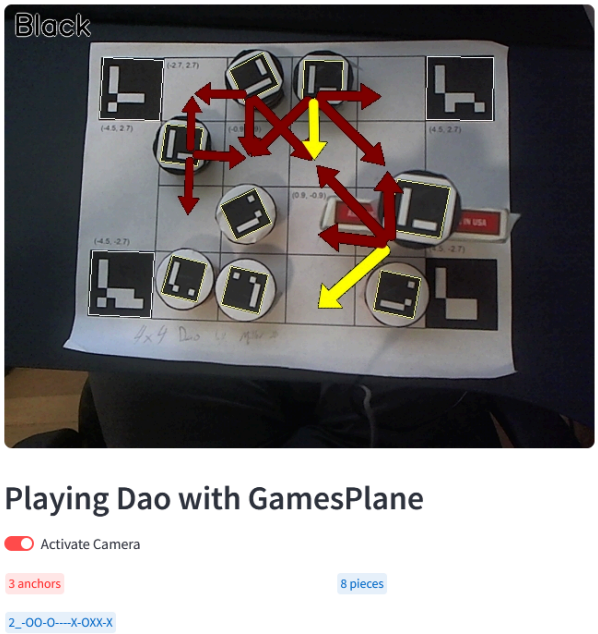

Figure 5.2. A deck of cards supports the middle-right black piece, changing its height compared to the other pieces. However, the value moves are still correctly displayed.

5.4: Piece Rotation Tolerance

Pieces may be rotated perpendicular to the plane of the board without impacting accuracy. In fact, testing with rotating pieces along other axes reveals GamesPlane is tolerant to these changes as well as demonstrated by Figure 5.3, though it may reduce the camera's ability to find the markers.

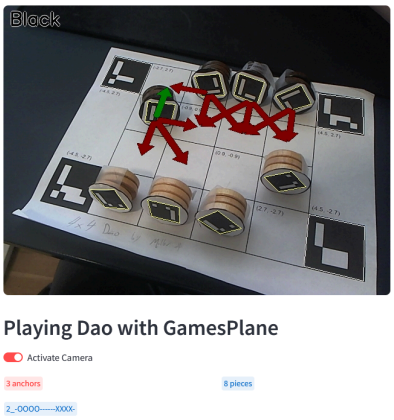

Figure 5.3. In the image, all pieces are rotated 90 degrees, but they are still detected correctly.

5.5: Poorly Aligned Piece

One failure case of GamesPlane is when a piece is placed close to an edge between two spaces. In Figure 5.4, the leftmost black piece is on the edge between two spaces, although its center is slightly towards the space on its left. The system incorrectly places it in the space on its right. In practice, this suggests that games with densely packed spaces could lead to drastically reduced accuracy.

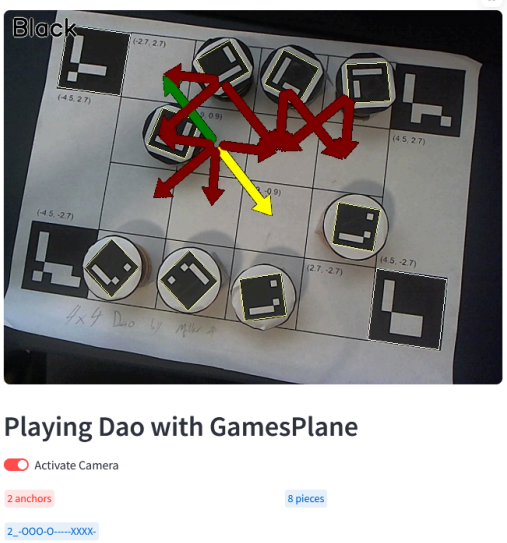

Figure 5.4. The leftmost black piece is not well-aligned to the center of the space it's meant to occupy. As a result, the detector believes it to be in the space to the right.

5.6: Camera Rotations

Rotating the camera can become a failure case depending on the axis of rotation. Rotations parallel to the plane of the board have no effect on accuracy as demonstrated by Figure 5.5.

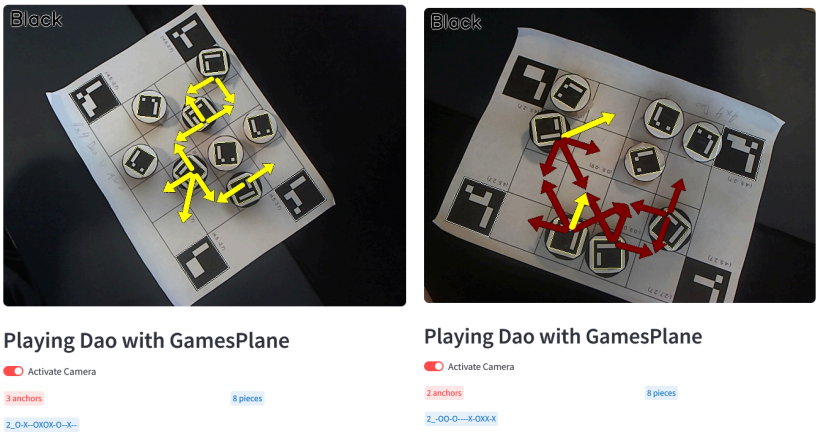

Figure 5.5. The first image has a 45 degree offset parallel to the board's plane. The second is nearly completely turned around. Testing either setup for 20 board states revealed no failures.

However, rotations on other axes that cause the camera to look at the board at a more sheer angle can impact accuracy (Figure 5.6), with the effect growing more pronounced at nearly-parallel angles until the ArUco markers fail to be detected entirely (Figure 5.7).

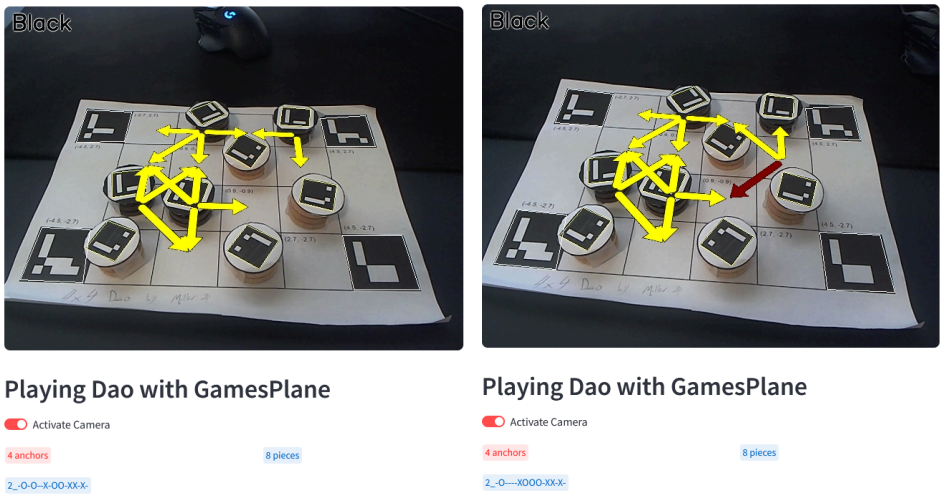

Figure 5.6. At a slight decline, the system sometimes flips rapidly between a correct and incorrect state. The two above states are examples of this phenomenon: the left is correct, the right is incorrect.

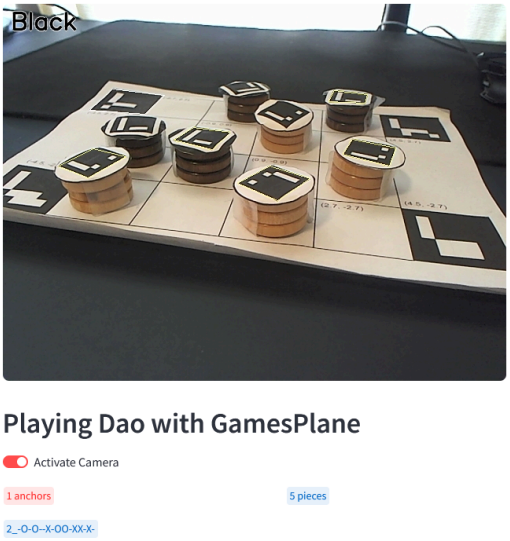

Figure 5.7. At a sheer angle, the markers fail to be detected entirely.

5.7: Other Shortcomings

There are a few other circumstances that can affect accuracy. For one, printed ArUco markers can curl at the edges, especially after long-term use. This lowers ease of detection at some angles. Also, if pieces do not have flat tops (such as chess pawns), the attached ArUco markers can be deformed by the uneven surface and potentially become undetectable.

GamesPlane renders moves on the plane of the board, not on the plane of the tops of pieces. When looking at the board from an angle in a game that has tall pieces, this can cause the value moves to be difficult to read as it's unclear which move refers to which piece. The issue is remedied from a top-down angle, however, as both the pieces and board are aligned.

Another consequence of ArUco-based design is that it can be hard to tell which pieces are which color. As the markers generally cover the entire top of the piece, it can become necessary to look at the board from a sideways angle to tell which pieces are which colors. Smaller ArUco markers could remedy this, but the smaller a marker is, the closer the camera must be to detect it.

5.8: Speed

The only limiting factor for GamesPlane's speed is the time taken to make the web requests to GamesmanUni that fetch board overlays. The web request usually takes about 2 seconds when GamesmanUni is run locally. It is worth noting that GamesPlane does not pause execution while fetching a web overlay — it instead displays the loading graphic shown in Figure 5.8 and continues running live, never breaking the AR illusion.

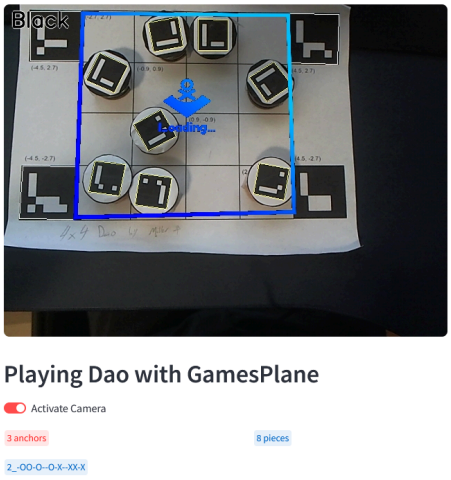

Figure 5.8. While loading a web request, a blue square is displayed so the player knows that the system is functional while they wait.

With a board state cached, the time between a state being visible in the camera and that state being displayed is practically imperceptible (under 0.1 seconds). GamesPlane also runs at high FPS (greater than 20) even with many markers in the frame on low-end hardware.

Chapter 6: Further Work

Thanks to its usability-based design, there are multiple routes for improving GamesPlane in the future.

Getting GamesPlane working on mobile devices is a natural idea. Augmented reality applications like GamesPlane traditionally run on phones, as it makes the AR effect more convincing: the camera is exactly where the screen is, so the superimposing effect looks more natural. Additionally, running the app on mobile phones would enable it to be used more widely by members of the general public.

GamesPlane would also benefit from being made available on GamesmanUni. One might imagine adding a “Play in AR” button to each game in GamesmanUni that would launch a GamesPlane session on the web. This would greatly improve public access to GamesPlane and skip over the local installation steps.

Alternate tracking methods could be another route for further development. Although Haar tracking was unsuccessful originally, the method is by no means impossible. It could possibly suffer from lower accuracy due to the less precise information garnered from Haar, but would allow for new piece designs.

In the vein of tracking, it would be interesting to test what sizes of ArUco markers function well with GamesPlane. The current markers cover their entire piece to maximize their size. However, with a sufficiently high-quality camera, smaller markers should be perfectly acceptable. The question of how small a marker can be and still be detected easily would improve the overall look of GamesPlane games.

Changing turns at the right time is essential for smooth play. Right now, turns change over as soon as a new board state is detected. However, with games that have moves which cover a lot of space, intermediate states may be incorrectly detected as final states, causing the turn to change before the player has completed their move. A better move detection system that takes movement over time into account would help avoid this issue.

These technical changes aside, the most likely kind of future improvements I expect to see are the addition of new games to GamesPlane. Thanks to its documentation and GamesPlane system, adding new games is by far the easiest improvement one can make to GamesPlane. Provided below is a list of games already on GamesmanUni that are highly compatible with GamesPlane. Any game not in this list has unusual pieces, move arrows that would be difficult to parse in AR, or another issue.

> 1D Chess, 4 Square Tic-Tac-Toe, Achi, All Bagh-Chai, Beeline, Change!, Chung-Toi, Connect 4, Dao, Dodgem, Dragons & Swans, Dōbutsu shōgi, Five Field Kono, Four Field Kono, Hare and Hounds, Ho-Bag Gonu, Jan, Joust, Kaooa, Kōnane, Lewthwaite’s Game, Mū Tōrere, Neutron, Nine Men’s Morris, Notakto, Nu Tic-Tac-Toe, Pong Hau K’i, Quick Chess, Slide-5, Tic-Tac-Toe, Tic-Tac-Two

Chapter 7: Conclusion

Altogether, GamesPlane successfully carves out a large niche for itself in the space of game state recognition. Although it requires the use of special ArUco markers, its speed, accuracy, and versatility make it an excellent tool for recognizing game states and viewing value moves. Especially considering the category of game it is built for — strategy games, where a single imperfection in the game state's recognition can lead to a completely incorrect suggested move — the tradeoff becomes especially worthwhile. Lastly, since boards can be printed out of paper and the system can be fully utilized without the need of a GPU to train a network, GamesPlane cements itself as a highly accessible tool, able to run even on low-end machines and to be physically reproduced at extremely low budget.

Beyond its practical use, with GamesPlane's modular design and extensibility, it is our hope that it will see continued development in future semesters. Board games played on a physical board with physical pieces are a wholesome source of joy for many, and so to be able to connect that love for play with the digital world is a great boon. This connection turns formidable topics such as web development, parallelism, or data compression into enjoyable ones. In the same way that learning to develop GamesmanUni can teach about web design in an engaging way, we hope GamesPlane is used as an entry into the intriguing world of computer vision and augmented reality by future GamesCrafters.

Bibliography

[1] S. Scher, R. Crabb, and J. Davis, "Making real games virtual: Tracking board game pieces," in *Proc. 19th Int. Conf. Pattern Recognit. (ICPR)*, Tampa, FL, USA, Dec. 2008. [Online]. Available: https://alumni.soe.ucsc.edu/~sscher/publications/ICPR2008-MakingRealGamesVirtual.pdf

[2] A. Mehta, "Augmented Reality Chess Analyzer (ARChessAnalyzer): In-Device Inference of Physical Chess Game Positions through Board Segmentation and Piece Recognition using Convolutional Neural Network," *arXiv preprint arXiv:2009.01649*, 2020. [Online]. Available: https://arxiv.org/abs/2009.01649

[3] R. Menezes and H. Maia, “An intelligent chess piece detection tool,” in *Proc. 1st Integrated Softw. and Hardw. Seminar,* Jun. 2023. [Online]. Available: https://sol.sbc.org.br/index.php/semish/article/view/25062

[4] M. A. Czyzewski, A. Laskowski, and S. Wasik, Chessboard and chess piece recognition with the support of neural networks, *Found. of Comput. and Decis. Sci.*, vol. 45, no. 4, 2020. [Online]. Available: https://sciendo.com/es/article/10.2478/fcds-2020-0014

[5] C. Belshe, “Chess piece detection,” Senior Project, Dept. Elect. Eng., California Polytechnic State Univ., San Luis Obispo, CA, USA, 2021. [Online]. Available: https://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1617&context=eesp

[6] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, “Automatic generation and detection of highly reliable fiducial markers under occlusion,” *Pattern Recognit.,* vol. 47, no. 6, pp. 2280–2292, Jun. 2014. [Online]. Available: https://cs-courses.mines.edu/csci507/schedule/24/ArUco.pdf

[7] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proc. IEEE Conf. Comp. Vis. and Pattern Recognit. (CVPR)*, Kauai, HI, USA, 2001, pp. I-511–I-518.

[8] D. Garcia, “GAMESMAN: A finite, two-person, perfect-information game generator,” Master’s Project, Dept. Comp. Sci., Univ. of California, Berkeley, Berkeley, CA, USA, 1995. [Online]. Available: https://people.eecs.berkeley.edu/~ddgarcia/software/gamesman/GAMESMAN.pdf

[9] S. Phyo, “GamesmanUni,” GamesmanUni. https://nyc.cs.berkeley.edu/uni/ (accessed May 20, 2025).

[10] M. Hollinger, “GamesPlane GitHub,” GitHub. https://github.com/MillerHollinger/GamesPlaneTesting (accessed May 20, 2025).

[11] E. Olson, "AprilTag: A robust and flexible visual fiducial system," 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, 2011, pp. 3400-3407, doi: 10.1109/ICRA.2011.5979561.

[12] G. Bradski, "The OpenCV Library," Dr. Dobb's Journal of Software Tools, 2000.

[13] Streamlit, “Streamlit – The fastest way to build and share data apps,” Streamlit.io. https://streamlit.io/ (accessed May 20, 2025).

Appendix: User's Guide

Appendix 1: Starting GamesPlane Locally

GamesPlane runs as a python-based web app. The following instructions cover how to start the app and begin using GamesPlane.

1. Ensure you have Python on your machine.
    a. GamesPlane was developed on Python 3.11.2, but higher versions will most likely function as well.
2. Clone the GamesPlane repository onto your machine.
    a. https://github.com/MillerHollinger/GamesPlaneTesting
3. Download the Python packages from the requirements.txt file.
    a. pip install -r requirements.txt (from the base directory)
4. Run the app from a console while in the base directory.
    a. `python -m streamlit run App/GamesPlane.py`

The app will automatically open in your default browser.
From here, go to the built-in starter guide to learn how to get started.

Appendix 2: Playing a Game with GamesPlane

Let's learn how to play an existing game with GamesPlane. You'll need:

- A GamesPlane board and pieces. You can find them in the `GamesPlane/App/PDFs` folder. Pick a game to play and print its board and pieces out. Use 8.5x11" printer paper and black ink.
- A camera. You can connect your phone camera to your computer or use a wired webcam. Webcams built into laptops can be hard to point at the board. Place the camera on a tripod or other holder so it can see the board from the top down. You can also just pick it up and point it at the board whenever you want to see an overlay; it doesn't need to be perfectly still.
- The GamesmanUni repository locally.

Now, follow these steps:

1. Check out GamesmanUni to the branch `ar/ae`. Build the site with `yarn build`.
2. Start GamesmanUni locally with `yarn dev`. Ensure it runs on `localhost:3000`.
3. Place the board on the table. Set up the pieces on it.
4. Point your webcam at the board.
5. Go to Launch Game. Click the game from the list.
6. Click "Activate Camera" to start your camera.
7. The screen should pop up with your camera's output. Move it around until it sees the board correctly.

At this point, it should display your AR video feed!

Appendix 3: Adding a New Game

Adding a new game to GamesPlane has these steps, which will be explained in more detail below:

1. Create a physical board and pieces
2. Create the `.json` file in Craftsman

3. Write a UWAPI converter
4. Register the game in `App/Launch Game.py`

Step 1: Create a Physical Board and Pieces
You'll need to create a board for your game with ArUco anchors, and then make ArUco pieces. You can start with this template: https://docs.google.com/presentation/d/1WUdp0DYTSKhTM55Ek5bQRoZIVrDISUIf8kJTF1hCyMw/edit?usp=sharing.

1. Make your board.
    - You must have at least one anchor printed on the board.
    - Each anchor should be a different ArUco ID.
2. Make your pieces.
    - Several pieces may share the same ArUco. It is common to do with with equivalent pieces: e.g. for checkers, all the black checkers could be ArUco ID 1 and all the white ones could be ID 2.
3. Print out your board and pieces.

You can click on an object and go Format options > Position to see its exact position. We often use From: Center in this panel to make the coordinate values smaller and easier to work with.

Step 2: Create the `.json` File in Craftsman
Go to the Craftsman tab in GamesPlane.

1. Add your game's name.
2. Set your Centimeters to Space. We usually use inches as space units, so type 2.54 unless you have a plan.
3. For each anchor, click Add New ArUco.
    - Its tag can be anything. We usually use `anchor_ID` where ID is the ID you will assign it.
    - ArUco ID is the ID of the specific anchor ArUco you are adding. Check the last page of the template for a key.
    - Size is the length of a black edge of the ArUco. You can use Format options to see the element's size in Google Slides.
    - Check "Anchored."
    - Set its position X and position Y to the values in Format options, i.e. its position in inches.
    - Click Add ArUco. Repeat for each anchor (4 anchors if you use the template)
4. For each piece type, click Add New ArUco.
    - Its tag should be a short name for how you refer to that piece. `"black"` and `"white"` are common choices.
    - ArUco ID is the ID of the specific piece ArUco you are adding. Check the last page of the template for a key.
    - Size is the length of a black edge of the ArUco. You can use Format options to see the element's size in Google Slides.
    - Click Add ArUco. Do not click Anchored.
5. Add the valid board positions. If you added spaces, you can again use Format Options to see the coordinates of each space you added.
6. (Optional) Upload the PDF of your game. You can also share it through other means.
7. Click "Save". The data you input will be saved to a `.json` file. You can edit that file if you change your mind on anything instead of retyping everything into Craftsman.

Step 3: Write a UWAPI Converter

You need to provide logic to convert a list of `DigitalAruco` objects to a UWAPI string.

1. Check the GamesmanUni of the game you are adding to understand how it forms UWAPI strings for that game.
2. Go to `App/UwapiConverter.py`. Create a class extending `UwapiConverter`; you can use `DaoConverter` as an example.
3. Give it the function convert. Again, see `DaoConverter` to see how this looks. It takes in the current turn and a list of `DigitalAruco` objects from pieces.
4. Write logic that uses the `DigitalAruco` objects (likely using their `.closest_board_position` property and `.tag` property) and the current turn (black or white) to make a UWAPI string.
    a. Not all board states are valid: You can return `"fail"` if the provided state should not be sent to GamesmanUni.

See below an example of what a `UwapiConverter` class looks like in Python.

```
class DaoConverter(UwapiConverter):
      # The convert() function is the critical function used to convert the
current turn and DigitalAruco piece list to a board state.

      @classmethod
      def convert(cls, turn: str, pieces: list[DigitalAruco]) -> str:
      if len(pieces) != 8:
            return "fail"

      # The passed-in turn is always “Black” or “White”.
      board_str = "2_" if turn == "Black" else "1_"

      # We map the various positions pieces may be into grid positions in
the UWAPI string.
      POS_TO_IDX = {
            2.7 : {
                  2.7 : [3, 0],
                  0.9 : [3, 1],
                  -0.9 : [3, 2],
                  -2.7 : [3, 3]
            },
            0.9 : {
                  2.7 : [2, 0],
                  0.9 : [2, 1],
                  -0.9 : [2, 2],
                  -2.7 : [2, 3]
            },
            -0.9 : {
                  2.7 : [1, 0],
                  0.9 : [1, 1],
                  -0.9 : [1, 2],
                  -2.7 : [1, 3]
            },
```

```
            -2.7 : {
                  2.7 : [0, 0],
                  0.9 : [0, 1],
                  -0.9 : [0, 2],
                  -2.7 : [0, 3]
            }
      }

      GRID = len(POS_TO_IDX)
      board_rep = [["-" for c in range(GRID)] for r in range(GRID)]

      # We iterate each DigitalAruco piece, converting it to a character
representing its color. You might use the .phys.id or the .tag to do so.
      for piece in pieces:
            name = "O" if piece.phys.id == 1 else "X"
            pos_x, pos_y = piece.closest_board_position
            idx_x, idx_y = POS_TO_IDX[pos_x][pos_y]
            board_rep[idx_x][idx_y] = name

      # Convert the grid to a string.
      for y in range(GRID):
            for x in range(GRID):
                  board_str += board_rep[x][y]

      return board_str
```

Step 4: Register the game in `App/Launch Game.py`
Now it's time to make `Launch Game.py` aware of your new game, which will add it as an option to play.

1. Find the dictionary `NAME_TO_INFO` near the top of the file.
2. Add your game to the dictionary in this format (you may use Dao as a reference again):
    a. `"Name" : {"route" : "the name of this game's route on GamesmanUni", "converter" : YourGamesConverter}`
    b. `"route"` forms part of the GamesmanUni URL. Here are some examples:
        i. `https://nyc.cs.berkeley.edu/uni/games/foxandhounds/variants/regular -> "foxandhounds"`
        ii. `https://nyc.cs.berkeley.edu/uni/games/fourfieldkono/variants/standard -> "fourfieldkono"`
        iii. `https://nyc.cs.berkeley.edu/uni/games/mutorere/variants/regular -> "mutorere"`
        iv. `"converter"` maps to your converter class. It is not a string.

Step 5: Play Your New Game!
Your game should now be added! Restart Streamlit if necessary and click your game's name in the list to play.

Appendix 4: Troubleshooting
Restarting Streamlit

This tends to fix a lot of the issues I run across: End your Streamlit session with CTRL+C in the console you used to start it. Wait a moment, then start it again.
I recommend closing old Streamlit windows as they can lag out your machine. You only need one Streamlit window open.

The game I want to play isn't in the list
It's possible the game simply hasn't been added to GamesPlane yet, or it wasn't added to the dictionary of games. Check with whoever added it.

The console says "Trying to access given camera..." a million times
This probably means your camera is in use, or hasn't yet finished up from the last time you ran the program.

Exit out of Streamlit with CTRL+C.
Close any programs that might be using your webcam, like Zoom, Google Meet, the Camera app, etc.
Wait 15 seconds.
Restart Streamlit by running the command again.

My video feed doesn't appear on Launch Game
Your camera is likely not being grabbed correctly because you have several cameras on your computer. Do this:

Go to the file `LaunchGame.py` in `App/Pages`.
Go to the line reading `ses.camera = cv2.VideoCapture(0, cv2.CAP_DSHOW)` (around line 46)

Now: Try changing the line to the following and rerun the page.

If you have a newer camera: `ses.camera = cv2.VideoCapture(0)`
If you have two cameras: `ses.camera = cv2.VideoCapture(1, cv2.CAP_DSHOW)`
If you have two newer cameras: `ses.camera = cv2.VideoCapture(1)`

No overlay is shown
Check if it's a winning state. Winning states don't have a GamesmanUni overlay, so nothing shows.
Check that anchors and pieces are visible. Anchors and pieces are outlined in white or yellow when detected. If they're not showing up, the game may be set up wrong, or the camera may be out of focus.

The AR loading overlay turns gray
This happens when the web request fails.
You might be on the wrong version of GamesmanUni -- you must be on the ae/ar branch. It also can happen if the yarn server is too slow.
Try closing out of the Streamlit app, then GamesmanUni. Then restart GamesmanUni and Streamlit in that order.
If this happens consistently and you are on the ae/ar branch, try this: Go to `App/BoardFetcher.py` and go to the line reading `await asyncio.sleep(2.5)`. This gives GamesmanUni 2.5 seconds to find the overlay. Change out 2.5 for a larger number (maybe 5 seconds) to give GamesmanUni more time to create the overlay.

The AR output isn't correct / shows the wrong moves

Check the turn. It's possible you have the turn set wrong. Click the turn switch to change the turn.

Check the ArUcos are correct. It's possible an ArUco is the wrong ID.

Check the ArUcos are the right size. If you accidentally printed anything too large, it will likely be detected incorrectly.

Straighten the camera. The camera should be facing directly down at the board.

I tried everything and it still doesn't work

Let me (Miller Hollinger) know at hollingermiller@gmail.com! There are a lot of moving parts in GamesPlane and I'm happy to help you fix them. You can also contact me on the GamesCrafters Slack.