AR Chess Advisor

FINAL REPORT

Team Number 05

Dr. Zambreno — Faculty Advisor & Client
Dillon Peters — Team Lead
Parker Bibus — Computer Vision Lead
Jake Aunan — AR Glass Lead
Jamie Peterson — Mobile Lead
Brett Santema — Testing Manager
Aidan Sherburne — Report Manager

Team Email: sdmay21-05@iastate.edu
Team Website: https://sdmay21-05.sd.ece.iastate.edu

Executive Summary

Strategy board game enthusiasts are constantly looking for ways to improve their skills at these games. Currently, their options are limited to reading books or articles on game strategies, learning from skilled professionals such as chess Grandmasters, playing more games against peers and bots, playing situational puzzles, or attempting to learn recommended moves from a game engine. These current methods are not only time consuming but also lack the levels of engagement and excitement required, especially by younger players.

Our solution is to develop a system that uses augmented reality (AR) glasses to take pictures of the game board and, using computer vision algorithms on the backend, determine the game board's state and pass it to a game engine for analysis. Finally, we will deliver a recommended move to be displayed on the AR glasses in real-time. The AR glasses will be responsible for capturing images of the chessboard and transferring these images to the paired Android mobile device. The backend will then perform the required image processing and, using computer vision algorithms, will determine the game state. The game state will then be communicated to an existing game engine implemented by the backend, and the resulting recommended move will be sent back to the AR glasses to be displayed to the end-user.

Development Standards & Practices Used

Digital Design Standards:

- Accessibility
    - Easy to use for the layperson who may not be familiar with AR Glasses and other advanced technologies
    - Colors allow for usage by colorblind individuals
- Mobile/Android
    - Clear layout and easy navigation
    - Flexible and modular components allowing for future expansion
    - Responsive, adaptive, and iterative
- Human-Centered - ISO 9241-210 [7]
    - Make interactive systems more usable by focusing on the use of the system and applying human factors and usability knowledge

Software Development Standards/Practices:

- Documentation
    - User documentation through user stories, feedback, and stakeholder reviews
    - Technical documentation through weekly and bi-weekly reports and Git
- Single Accessible Repository – Git and GitLab
- Code

    - Style
    - Modularity
    - Naming Conventions
    - Comments
- Agile Methodology Standards
    - Respond to change rather than following a strict plan
    - Collaboration with the customer
    - Prioritize working software over documentation
    - Developing Information for Users in an Agile Environment – IEEE 26515 - 2018[6]
- Lean Development – fail-fast
    - Release a series of minimum viable products (MVPs) to learn what users do and do not like and iterate.
- Software Quality Assurance Process Standards for Testing – IEEE Std 730-2014 [4]
- Software Reviews – IEEE Std 1028-2008 [5]

Engineering Standards:

- Reliability
    - Accuracy thresholds > 95%
    - Crash Rates < 1%
- Scalability
    - Easy to package and distribute the application
- Performance
    - Response times allow for undisrupted play
    - Minimize battery life impact

Summary of Requirements

Functional Requirements:

- An augmented reality (AR) glasses device that is nonintrusive (buy)
- Capture the game board state using the AR glasses
- Detect and process the board state using computer vision (CV) algorithms
- Determine the best move given the current state using top game engine(s)
- Indicate recommended moves to the user on the AR glass display

Environmental Requirements:

- The strategy board game should be played in a well-lit room
- For best results, the area around the gameboard should have sharp contrast
- The AR device should be kept in a clean, dry, and dust-free environment
- The strategy board game should be played in a low dust environment to keep the lens clear

Economic Requirements:

- The project should not exceed $1000 in cost. Advisor approval is required for expenditure exceeding $300.
- A proof-of-concept product should be completed no later than the end of Spring Semester 2021
- "Build when you can, buy when necessary"
- Android smartphones may be necessary to serve as an emulator until the AR glasses are purchased

Applicable Courses from Iowa State University Curriculum

Iowa State University courses with content applicable to our project include:

- CPR E 185: Intro to Problem Solving I
- CPR E 186: Intro to Problem Solving II: Project
- CPR E 230: Cyber Security Fundamentals
- COM S 227: Object-Oriented Design
- COM S 228: Data Structures
- COM S 309: Software Development Practices
- COM S 311: Introduction to Algorithm Design and Efficiency
- CPR E 308: Operating Systems, Principles, and Practice
- CPR E 388: Mobile Platforms
- ENGL 314: Reporting, Documenting, and Technical Communication
- S E 319: Construction of User Interfaces
- S E 329: Software Project Management
- S E 339: Software Architecture and Design
- CPR E 530: Network Protocols and Security

New Skills/Knowledge acquired that were not taught in courses

- Requirements Development
- Engineering Standards
- Encoding and Decoding of Images
- Running an Executable in the Background of a Mobile Application
- Basic Computer Vision with OpenCV
    - Line detection
    - Point detection
    - Pre-processing techniques
    - Color Masking
- Jupyter Notebooks for Iterative Development
- Running Python Code on Android
- Running executables in the background of an android application and giving said executable commands
- Learning to speak and present a technical project professionally

Table of Contents

List of figures/tables/symbols/definitions

List of Figures

List of Tables

No table of figures entries found.

1 Introduction

1.1 Acknowledgment

The team would like to thank the Iowa State University Department of Electrical and Computer Engineering for giving us resources, guidance, and expert consultation. We appreciate the Electronic Technology Group for providing us with our team website, server resources and ordering the augmented reality glasses and chess set for our project. We would also like to thank Dr. Joseph Zambreno for meeting with us weekly to give us guidance and advice while serving as the Product Owner/Client.

1.2 Problem and Project Statement

Strategy board game enthusiasts are continually looking for ways to improve their skills at these games. Currently, their options are limited to reading books or articles on game strategy, learning from skilled professionals such as chess Grandmasters, practicing by playing games against peers and bots, playing situational puzzles, or following recommendations given by a game engine.

These current methods are not only time consuming but also lack the levels of engagement and excitement required to make them worthwhile, especially for younger players.

Our solution is a system that uses augmented reality (AR) glasses to analyze the state of the game board and game pieces. Using computer vision algorithms, we determine the game board's state and pass it to a game engine for analysis. Finally, we deliver a recommended move to be displayed on the AR glasses in real-time. The AR glasses are responsible for capturing images of the game board, processing them, and, using computer vision algorithms, determining the game state. The game state is then communicated to a game engine, and the resulting recommended move is sent back to the AR glasses to be displayed to the end-user.

Users of our application can get move recommendations from a game engine or build skill by playing games against a computer controlled by a game engine.

1.3 Operational Environment

The AR glasses used for capturing images and displaying the recommended move should be stored in a clean environment free of dust and any objects that may scratch the camera or glass lenses. When using the AR glasses, the user should be in a well-lit room where the area around the board is in sharp contrast to allow for consistent board and state detection.

The corresponding backend running the game (chess) engine may in the future live in a separate location and would need to communicate with the AR glasses through WIFI such that it can connect to the AR glasses to receive the image transmissions and communicate recommendations between devices. We assume this communication connection is autocompleted upon powering up the AR glasses after the initial pairing synchronization.

The battery life and heat of the headset are also a concern. Given the high CPU usage necessary to do image processing, keeping the glasses powered is a concern. To limit this concern, we recommend that the user charge the glasses after extended periods of use or use the glasses near

outlets that allow you to charge the glasses during operation. Additionally, high CPU usage will generate a decent amount of heat. Therefore, we recommend you use the glasses in a cool, stable, temperature environment such as one's own home. Additionally, we recommend avoiding using the sunglasses in the direct summer heat as the sunlight will increase the temperature of the Vuzix Blade device and increase the likelihood of temperature caused power off.

1.4 Requirements and Standards

Functional Requirements:

- An augmented reality (AR) glasses device that is nonintrusive (buy)
- Capture the game board state using the AR glasses
- Detect and process the board state using computer vision (CV) algorithms
- Determine the best move given the current state using top game engine(s)
- Indicate recommended moves to the user on the AR glass display

Environmental Requirements:

- The Strategy board game should be played in a well-lit room
- For best results, the area around the gameboard should have sharp contrast
- The AR device should be kept in a clean, dry, and dust-free environment
- The Strategy board game should be played in a low dust environment to keep the lens clear

Economic Requirements:

- The project should not exceed $1000 in cost. Advisor approval is required for expenditure exceeding $300.
- A proof-of-concept product should be completed no later than the end of Spring Semester 2021
- "Build when you can, buy when necessary"
- Android smartphones may be necessary to serve as an emulator until the AR glasses are purchased

Digital Design Standards:

- Accessibility
    - Easy to use for the layperson who may not be familiar with AR Glasses and other advanced technologies
    - Colors allow for usage by colorblind individuals
- Mobile/Android
    - Clear layout and easy navigation
    - Flexible and modular components allowing for future expansion
    - Responsive, adaptive, and iterative
- Human-Centered - ISO 9241-210 [7]
    - Make interactive systems more usable by focusing on the use of the system and applying human factors and usability knowledge

Software Development Standards/Practices:

- Documentation
    - User documentation through user stories, feedback, and stakeholder reviews
    - Technical documentation through weekly and bi-weekly reports and Git
- Single Accessible Repository – Git and GitLab
- Code
    - Style
    - Modularity
    - Naming Conventions
    - Comments
- Agile Methodology Standards
    - Respond to change rather than following a strict plan
    - Collaboration with the customer
    - Prioritize working software over documentation.
    - Developing Information for Users in an Agile Environment – IEEE 26515 - 2018[6]
        - This standard is used as a basis for how we will manage information throughout our Agile Development
- Lean Development – fail-fast
    - Release a series of minimum viable products (MVPs) to learn what users do and do not like and iterate.
- Software Quality Assurance Process Standards for Testing – IEEE Std 730-2014 [4]
    - This standard is used as the basis for our testing program
- Software Reviews – IEEE Std 1028-2008 [5]
    - To be used in our Level 2 Acceptance Testing, however, we did not reach this level of testing

Engineering Standards:

- Reliability
    - Accuracy thresholds > 95%
    - Crash Rates < 1%
- Scalability
    - Easy to package and distribute the application
- Performance
    - Response times allow for undisrupted play
    - Minimize battery life impact

1.5 Intended Users and Uses

Intended User:

This solution is intended to be used by consumers that need assistance with succeeding in certain tabletop games (focus on chess). This could include:

- Novices looking to defeat experienced players

- Players looking to learn and improve at the game through constant advice
- Players who want to play against a game AI stored in a set of smart glasses

Intended Uses:

1. The user wants to play against the application's AI
    1.1. The user sets up the board and begins the game
    1.2. User lines up the board with the camera and scans for his chosen opponent color
    1.3. The program examines the board and gives a visual cue for advice on how the given opponent should move
    1.4. The user performs the physical move for the AI (Artificial Intelligence) and repeats until the game ends
2. The user wants to scan a board to get advice on his next move
    2.1. The user opens the application on their smart glasses and activates the computer vision component
    2.2. The user lines the board up with the camera and takes a picture to be scanned
    2.3. The program examines the board and gives a visual cue for advice on the user's next move
    2.4. The user performs the move on the board

1.6 Assumptions and Limitations

Assumptions:

- The game is being played in a well-lit room. Good lighting is necessary for computer vision to work properly. Ideally, the game board should have a high contrast background, and the game should be played in a low-dust environment. Also, the lens on the AR glasses should be kept clear and dust-free.
- The game board and pieces are traditional, and rule standards are followed. For example, a game of chess is being played on a standard 8x8 board with the standard ruleset as established by FIDE.
- It is assumed that the players are using standard/tournament game pieces. This is so that the computer vision can consistently identify the game pieces and not have to accommodate several different variants of pieces.
- The person wearing the AR glasses is the person playing the game and not a spectator. We are choosing to focus on the player aspect to remain true to the original scope of the project, which is to have the AR glasses suggesting moves directly to the player.
- The chess board has red dots in each square and green dots on each of the four board corners.

Limitations:

- The project should not exceed $1000. The budget is to be used primarily for the Vuzix blade glasses, which have already been ordered.
- Minimum Viable Product should be completed no later than the end of spring semester 2021

- The app may not be able to capture the game state accurately mid-game. The computer vision cannot account for: knowing if a player can castle, knowing whose turn it is, and detecting a draw via 3-fold repetition. Additionally, we do not want the user to have to enter any of the game state information manually. Therefore, we are limited to only starting at the beginning of a game, and the user will not be able to always get accurate moves if they attempt to put the glasses on mid-game.
- The user will not be able to play with a heavy time restriction (e.g., bullet chess) due to the time needed for calibration processing. The game engine API (Application Program Interface) allows us to set a maximum time to calculate a move. Still, additional time is needed for computer vision processing when calibrating the pipeline.

1.7 Expected End Product and Deliverables

Our project will consist of three main pieces: An AR glasses front end, a computer vision back-end system, and a strategy board game engine back-end system. These three items will be present in each of our deliverable stages as they advance towards the final product. Our project will have several key deliverable stages broken up into:

1. Finalized approach – September 18, 2020

The finalized approach is where our project will lock in the technologies that we plan to use. There are a multitude of options for AR glasses and game engines to choose from, and decisions need to be made to advance the project. After the research is complete and technology is chosen, we will architect the project and create a development schedule. At this point, the team is ready to start development with a solid plan in place for moving forward.

2. Proof of Concept – End of Fall 2020

The proof of concept is a showcase of our technology at work. At this stage, we have our AR glasses able to display basic information, get images, and send images. Our computer vision can recognize the game board. And our game engine will be integrating with the android app. The three pieces of the project may not all work together yet to play the strategy board game, but they are starting to come together. AR glass can send a game board image, computer vision can recognize items on a game board, and the game engine is being integrated.

3. Minimum Viable Product – March 2021

The minimum viable product will be delivered when everything is working together to assist at playing the strategy board game. In this stage, a user will be able to put on the AR Glasses, look at a game board (at least when starting in the opening state) and receive recommended moves. AR glass UI may not have final polish, and computer vision may not be perfect at recognizing the board in some states, but the initial sequence of the match of a strategy board game can be played with assistance.

4. Final Design – End of Spring 2021

The final design will take the minimum viable product and add polish. UI (User Interface) will be smooth and clean; computer vision will be able to recognize game pieces and board clearly

and accurately. At this stage, someone who has never played the strategy board game or used AR glasses before should be able to put on our product and competently play a game.

2 Project Plan

2.1 Project Tracking Procedures

Our group is utilizing several different tools to track progress and ensure effective communication. Our primary method of staying connected day-to-day is a GroupMe chat that is used for informal communication and quick, non-essential updates on work. This is meant to be used to keep us all in touch and on the same page, but not used for major discussions or posts that can be lost easily. Our team has two weekly meetings for important discussions: one with our advisor, Dr. Zambreno, and one without. These meetings are used to go over our progress in the days between meetings, discuss upcoming work, and complete group work. Meeting notes are stored on a shared Google Drive that we use for all important documentation. This includes research, notes, assignments, and design information. Additionally, our group used GitLab to store and track our progress on development work over time. Lastly, our group used Trello boards for monitoring tasks during our sprints. While our initial development was more waterfall, this was necessary to get the project off the group, and we transitioned entirely to the agile methodology after the initial baseline was created. Regardless of what methodology (or mix) we are currently using, the Trello board will be significantly easier to track tasks than google drive.

2.2 Financial Requirements

To conduct this project, we purchased the Vuzix Blade AR Glasses. They were purchased on sale through Amazon for $499.99, pre-tax. Additionally, we purchased a traditional chessboard and piece set from Amazon for $19.99, pre-tax.

2.3 Other Resource Requirements

The physical resources needed to complete this project were:

- A chessboard (gameboard) and chess (game) pieces: We purchased a traditional chess set off amazon. This set will ideally have the lettering and numbering such that a user unfamiliar with chess knows where a location on the board, such as F6, is.
- Physical space for testing and development as needed: As the project progressed, there was more interaction between members of the team as components were integrated and when testing was performed. This physical space was the TLA in Coover Hall.
- Android Development Environment: The team needed an environment to develop the AR applications. The Android Studio and its emulators were used for this purpose.

Additionally, we required additional knowledge resources in the form of conversations with expert faculty.

3 Design

3.1 Previous Work and Literature

Our project is similar to existing chess engine mobile applications. These applications consist of a chess GUI and engine that allow a user to play chess against an AI with top ELO chess engines' assistance. There are a variety of applications across many platforms, but the most famous one is the Android application known as DroidFish [1]. We are looking to perform the same backend functionality as this app, but instead of playing the game on the phone, we used AR glasses to analyze a physical board. While our app will not have near the customization possible with DroidFish, our project's advantage is the expansion into physical game analysis through AR. By expanding the project into the real world, we add a computer vision element into the project that is based on a variety of published papers and methodologies.

When it comes to the computer vision aspect of our project, there has been a good bit of research, and even some projects, whose aim is to recognize a chessboard and the piece positions. The two major methods used for recognition are feature recognition/machine learning and trained neural networks. The main recognition technique observed for detecting the chessboard itself has been feature recognition, specifically techniques using Hough Transform Line Detection with data processing on the lines found. However, some methods do use trained neural networks to do chessboard detection. When it comes to the actual piece detection, the most used technique seems to switch, with neural networks becoming a more present way of determining the chess piece. The two main projects that we have used in figuring out the techniques we want to use, and with just testing different things in general, have been the ChessboardDetect by Elucidation [2] and Visual Chess Recognition by Danner and Kafafy [3].

3.2 Design Thinking

Given our project's nature, the empathize and define stages of design thinking had already been completed by our product owner, Dr. Zambreno. He explained his thoughts and conclusions from these phases in our initial kickoff meeting. Initially, Dr. Zambreno defined the problem to focus solely on a chess implementation, but after further consideration and discussions with our team, we expanded our project to be more modular and allow for flexibility and easy implementation of multiple strategy board games. Our problem definition can be found above in section 1.2.

Using this problem definition, we moved to the ideate state. The nature of how our problem was defined did not allow for a lot of design flexibility regarding how to attack the problem, but rather what components, libraries, and games we should pick. In this stage, we explored a variety of different AR Glass options, including major names such as the Google Glasses and Microsoft HoloLens. In conjunction with the AR Glass ideation, we investigated a variety of computer vision techniques and libraries, including OpenCV, Computer Vision Toolbox, ML.Net, and Google Collab. We also explored a variety of options for games to implement, including tic-tac-toe, connect-4, checkers, and chess. We felt that we should shoot for a chess implementation, but we wanted to have other more straightforward options if it would prove difficult. After selecting chess as our game of choice, we began researching existing Android and iOS apps that were similar as well as powerful chess engines such as Stockfish, Komodo, and Cuckoo. We then took our six ideas

for AR Glasses and chess engines. We narrowed them down to one idea each before presenting these recommended options to our Product Owner, Dr. Zambreno. After receiving Dr. Zambreno's approval, we began fleshing out a prototype development plan.

3.3 Initial Design

Our current approach can be broken down into three areas: AR Glass Application, Computer Vision, and Game Engine. The flow of our approach is:

1. Take Calibration Images using chessboard pattern
2. Send Images to Computer Vision Module to calibrate the OpenCV algorithms
3. Send Okay Signal to AR Glass application from backend indicating ready for use
4. Take a snapshot of the chessboard using AR Glasses
5. Send snapshot to the computer vision module
6. Run computer vision algorithm to determine the chessboard state
7. Pass the board state to the game engine to determine the recommended move
8. Send recommended move to the AR Glass Application
9. Display Recommended move on the AR Glass Application
10. Repeat Steps 4-9 as the user continues to play

Each module of our project is broken down below:

AR Glass Application:

- Using the Vuzix Blade's camera, the application will take a picture of the chessboard and pass this image to the backend, which may live on a paired mobile device.
- The camera on the Vuzix will need to be calibrated upon first-time use using standard OpenCV calibration algorithms.
- If the backend lives on the mobile device, the Vuzix Blade will need to be connected to WIFI and will share the image via HTTP server calls, or connected to the device through Bluetooth, and the image will be sent directly between the paired devices.
- The Glass application will also be responsible for displaying the chess engine's recommended move on the AR display.
    - Ideally, this recommended move would be overlayed on the chessboard; however, we plan to display the recommended move as text using proper chess terminology.

Computer Vision:

- The computer vision processing for our application will run on the "backend" application.
- Upon receiving the image on the mobile device, we will run computer vision algorithms using OpenCV to determine the current state of the chessboard.
- Setup algorithms to ensure calibration.
- Some image pre-processing may occur on the Vuzix Blade device, depending on its computation abilities.

Game Engine:

- The game engine will run on the "backend."
- The game engine is responsible for calculating the best move from the current board state as determined by computer vision.
- After determining the recommended move, the game engine module will need to communicate the move to the AR Glasses.
    - The game engine output will need to be converted to something that the layperson can understand.

- The engine should be customizable such that users can determine how many moves to analyze before providing a recommendation.
- The engine development should be extremely modular and allow for usage of multiple engines for multiple types of games as needed (Ex: Connect-4, chess, and checkers)

Ideally, the AR Glasses will host the front end and back-end applications, however, the backend may have to be hosted on a separate device due to processing limitations as increased functionality is added and accuracy is required. The three modules discussed above, along with the purchase of the Vuzix Blade AR Glasses, satisfy all functional requirements of our design. Additionally, the above design uses the Vuzix Blade, which was purchased for less than $500, meeting the purchase price requirement for this project. As part of this purchase, we are given access to a Vuzix Blade Developer account that provides emulator access, such that physical Android devices are not needed.

The above design also follows the "build when you can, purchase when necessary" philosophy we want to follow for this project. The design description above does not touch on any of the Operational Requirements described in section 1.4; however, a simple information sheet included in our first appendix and developer guidelines detail this.

3.4 Technology Considerations

When analyzing technology, we looked at three areas: AR Glasses, Computer Vision, and Game Engine. We found a variety of AR Glass options, however, due to budget and computing requirements, only three of these options were given any significant consideration:

- Google Glass Enterprise 2.0
    - Strengths: Lightweight, fast processor, supports voice commands, has a good camera, lots of support is available, and the development is Android-based, which we are familiar with.
    - Weaknesses: $1000 dollar price tag, emulator options are less than stellar
- Microsoft HoloLens (Edition 1 or 2)
    - Strengths: Comfortable, supports gesture and voice recognition, large FOV, is extremely powerful, and lots of support and resources available.
    - Weaknesses: Extremely expensive and is outside of our price range. However, we may be able to check this out from the department. The glasses are large and not discrete.
- Vuzix Blade
    - Strengths: Glasses are discrete, Developer account provides company support, supports voice and gesture commands, and the development is Android-based which we are familiar with.
    - Weaknesses: Not as powerful as the other two options, community support and examples are extremely limited.

We decided that the Vuzix Blade was the best option as it had all the features necessary to complete the project, was half the price of the Google Glasses, and was just as discrete.

While deciding on the hardware to use, we also did research into potential computer vision solutions. The two realistic solutions we found were OpenCV and MATLAB Computer Vision Toolbox. The strengths and weaknesses of each are as follows:

- OpenCV
    - Strengths: Open source, free, industry-standard, active community, large library of algorithms, multiplatform support, multilanguage support (including ones we know).
    - Weaknesses: Massive library, lots of guides used older versions, which may have some breaking changes (fixable but not always easy to find).
- MATLAB CV Toolbox
    - Strengths: Large library of CV tools, MATLAB language is made for data manipulation (what we are doing).
    - Weaknesses: Only usable in MATLAB, must have MATLAB program access, we do not have much experience with MATLAB.

With these strengths and weaknesses in mind, we decided to use OpenCV. The strongest reasons for this were the experience we had with OpenCV supported languages, strong community support, and the Open-Source availability. However, having done this research, we know that MATLAB CV Toolbox is a viable alternative that we can use if OpenCV does not work as well as we need it to.

We considered a variety of chess engines such as Stockfish, Komodo, Cuckoo, Leena Chess Zero, and Shredder. These engines all have similar capabilities, and therefore we picked the one that would be easiest to work with, which was Stockfish. This engine has precompiled binaries suited for Android, was the most well-documented engine, and had implementation examples that made it an easy choice.

3.5 Design Analysis

So far, our proposed design has worked. Dr. Zambreno encouraged us to separate our design from implementation wherever we can, resulting in our plan being very flexible and avoiding specific implementation details such as mentioning a specific library we plan to use. Additionally, we designed to accommodate the potential for the backend to live off glasses. Thankfully, this was not required, and we were able to run the computer vision and stockfish components within the android application.

The only change we have made so far has been in implementation, not design. This change was in the computer vision module. The initial method and algorithm we tried were not determining the lines and squares on the board at a level we would like, and we pivoted to another algorithm and method that has been more promising so far. This pivot is discussed in detail in the computer vision implementation section of this report.

Additionally, the project's requirements and client expectations did not change since the project was kicked off in CPRE 491. This allowed us to stick with our final design from CPRE 491, which is described in this section. As we progressed throughout the project, we did not get to some of the user testing we had hoped for such as with the ISU (Iowa State University) Chess Club. As a result, we did not get additional feedback and observations from these missed demos and tests. Therefore, we did not get to iterate on our implementation and design as we would have liked had we gotten to do these tests and demos.

3.6 Development Process

Our team decided to use an Agile development approach for our AR Chess Advisor project. By following this approach, we were able to manage our time best and ensure we are working efficiently towards our client's goals. Initially, our development did not appear to follow Agile practices due to the initial setup work that is necessary to support the iterative Agile methodology. However, after the first couple of weeks of initial setup, we were positioned to transition to a modified Scrum.

We had two scheduled meetings each week, where we gave status updates and planned what work needed to get done for the coming "sprint." One meeting each week included our client, allowing us to update them on our progress and receive regular feedback. This model allowed us to continually modify our tasks and pivot our focus areas as the project progresses to meet the client's needs best.

We continued to use the Agile approach throughout the duration of the project, though the weekly meetings went from having a greater focus on planning, researching, and reporting findings to have a greater focus on individual statuses, integrations, and demos.

3.7 Design Plan

Our system is relatively simple architecturally, as we host the front end and back-end components entirely on the AR Glasses. This greatly simplifies our design's architecture as the interaction among modules is all contained within the same device. However, the design does include a safety net of communication libraries and modules should the backend be hosted on a different device in the future.

This section will highlight figures representing the modular design, flow, and sequence diagrams that illustrate how the design components will interact. For a more detailed breakdown of these components, please consult the corresponding sections in Chapters 2 and 3

The team has designed the system illustrated below in Figure 1.

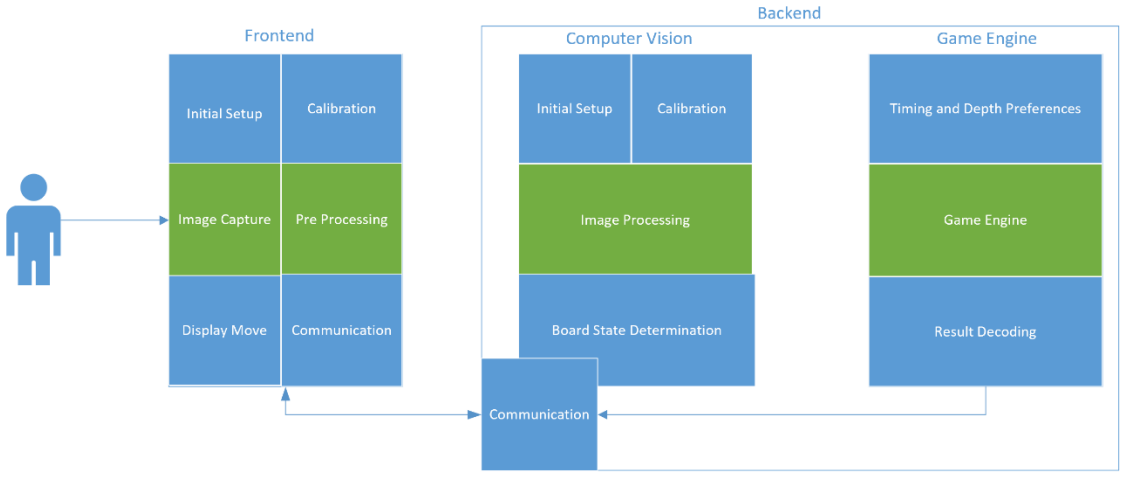


Figure 1: Modular Design Diagram

Each of the functional requirements outside of obtaining the AR Glasses is accomplished by some combination of the modules in this diagram:

- Capture the game board state using the AR glasses: This the responsibility of the front-end module. The front-end module will be responsible for capturing a usable image of the board state and doing any pre-processing on the image that is necessary for the computer vision module.
- Detect and process the board state using computer vision algorithms: Using the image captured from the front-end module, the computer vision module will take the image and transform it into a data structure to represent the board state using a variety of algorithms for the board, piece, and location detection.
- Determine the best move given the current state using top game engine(s): This is the responsibility of the game engine module. Using the board state passed from the computer vision module and the corresponding engine command, the engine should output the recommended move.
- Indicate recommended moves to the user on the AR glass display: Using the output from the game engine, the user should see the recommended move on-screen using language understandable to the layperson.

The components above are extremely modular and cater to both use cases: playing the game with assistance and playing against the engine. The flow for both use cases is similar, with the major difference being what color the engine believes you are playing as. Given the similarities and modularity, adapting to allow for both use cases will be remarkably simple.

A majority of the submodules have no significant constraints beyond their functional requirements. The exceptions are the computer vision submodules. These modules have the additional constraint of being relatively fast such that the user's game experience is not interrupted. This was only a problem for the calibration at the beginning of the game.

In addition to the system design sketch, Figure 2, below, shows the high-level flow through these modules and provides a visual representation of how the modules interact to satisfy the functional requirements described above.

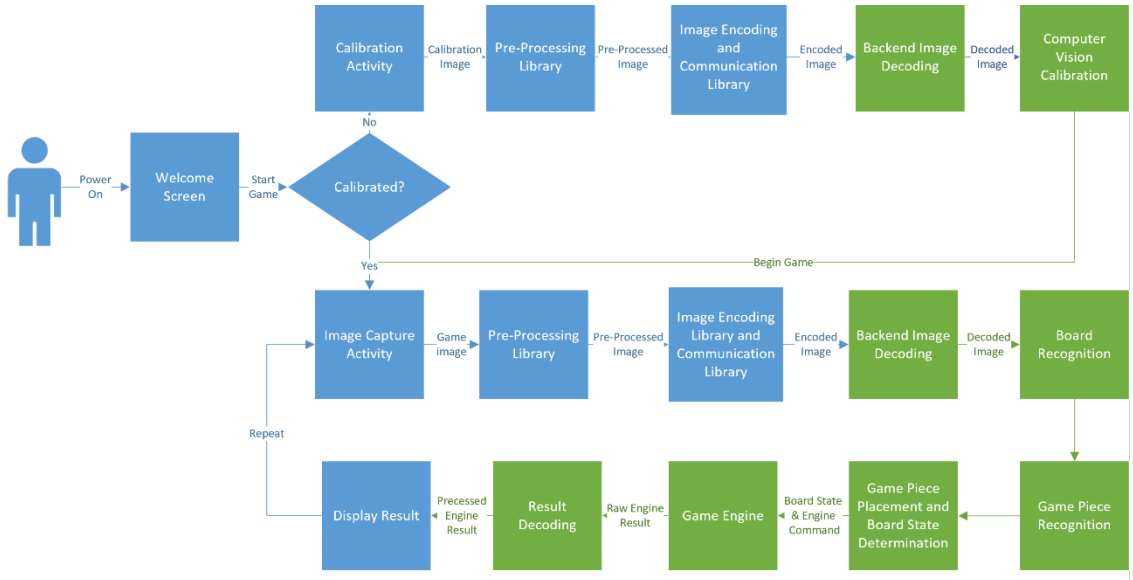


Figure 2: Component Flow Diagram

Since we have no initial pairing or connections to the backend to manage, the flow is greatly simplified. The only decision to be made is whether calibration of the computer vision algorithms is required. If so, we take the top branch; if not, the user is ready to play the game and can take the bottom branch. The bottom branch is a circular flow that repeats while the user is playing the game. In the above diagram, front-end components are in blue while backend components are in green. The diagram omits the shutdown sequence for if the user powers off the glasses as this is very straightforward.

An even higher-level functional sequence can be seen below in Figure 3.

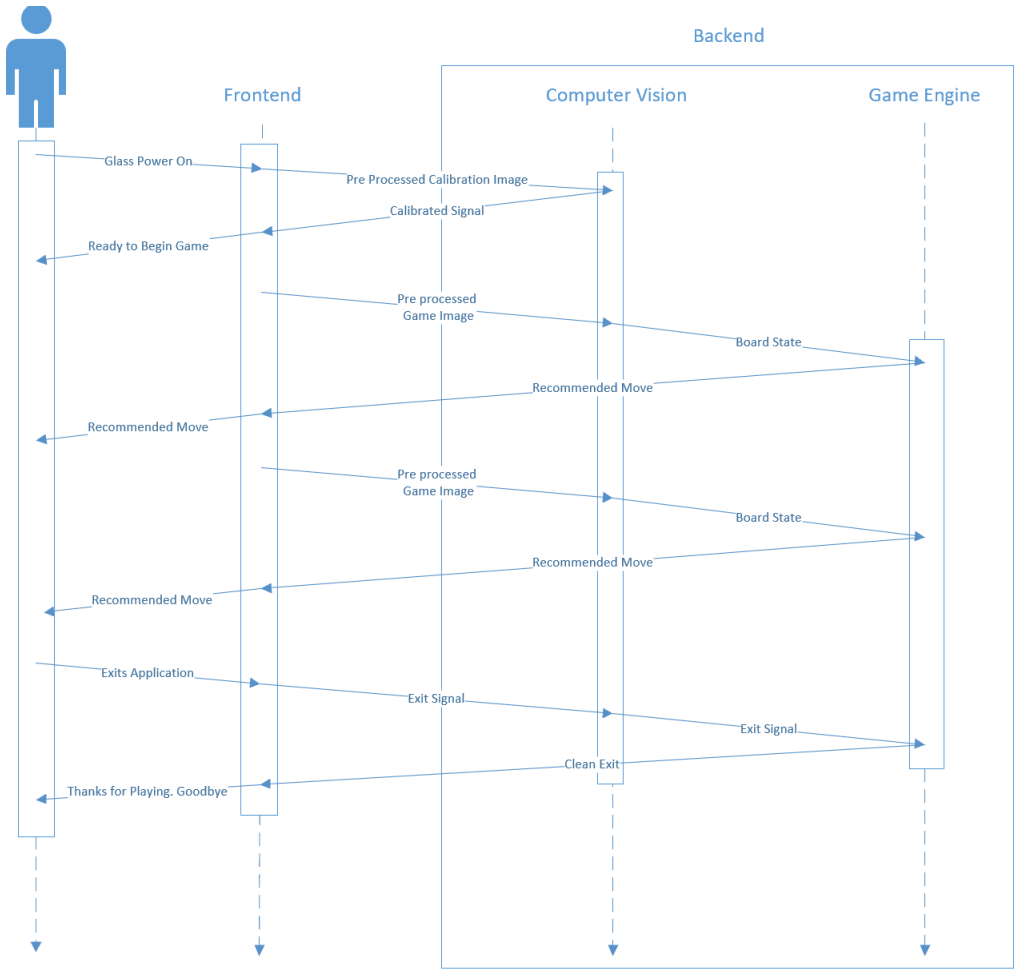


Figure 3: High Level Sequence Diagram

While the Vuzix Blade does run a modified Android operating system, there are unique navigation and display elements as well as size constraints that prevent us from developing specific UX screens for the device. However, all screens will adhere to the Vuzix Developer Account recommendations for both design and implementation and undergo team reviews and UX iterations as they are developed. In section 4: Implementation, we will have a few screenshots of implemented screens

according to the recommendations from Vuzix. The user interfaces for this project are enumerated below:

- Welcome Activity
- Gameplay Activity
    - Calibration Activity
- Initial Setup Activities (Not implemented by us, but were part of our initial design)

3.8 Design Changes in 492

Our design did not change in CPRE 492. Dr. Zambreno encouraged us to start development early, which we did in late September of CPRE 491, which allowed us to refine our design early in 491 such that modifications to the design were not required in CPRE 492. While design modifications were not required as part of 492, we did need to change our implementation of the computer vision module over winter break. Our initial approach was extremely complex mathematically and difficult to understand. This made it difficult for Aidan and Parker to progress, and therefore they reached out to different faculty at ISU for help. Based on the feedback received, Parker made the decision to start over on the implementation with some additional assumptions to get a working implementation first before iterating to make improvements. These changes will be discussed in detail in the implementation section of this report.

3.9 Security Concerns and Countermeasures

Due to the nature of our device being an application used for fun and not dealing with any serious data, the security concerns are extremely limited unless attackers gain camera access. Ignoring the camera aspect momentarily, the main attacks against our application would be related to relaying incorrect move or error information to the user. While this would compromise the core gameplay functionality of the Game Advisor application, the motive for carrying out such an attack is next to zero. The glasses are meant to be used for fun or as a learning tool, not in any sort of competition, as they would be considered cheating. Therefore, given the extremely low likeliness of this threat, we did not feel it was worth adding in security measures to protect against it.

When factoring in the camera usage of our application, the security risks increase. Compromising the application would allow the hacker to potentially see all that the user sees. In the case of our application, the user should be looking at or near a chessboard nearly all the time while wearing the glasses. Therefore, the camera information should not be super useful for that attacker. However, if the compromise of our application allows the hackers to remotely turn on and access the device and its camera at will, the information obtained depends on the glass's storage. Since we recommend that users store the glasses in their case, there should be nothing for hackers to see.

The other two areas of concern are privacy and physical security. The technology in the Vuzix Blade allows for advanced features such as tracking the user's eye movement. While our application does not utilize this feature, the device itself may be logging this information regardless, and attackers may wish to access it. The responsibility for security of this information and functionality we feel belongs to the device manufacturer. The most relevant security concern is the device's physical security. Like all mobile devices, it is easy for the device to be stolen or lost at which point all data on the device could be exposed. Like all other mobile devices, the responsibility for physical security lies with the user/owner of the device. While we would like to implement security

mechanisms to deal with the cyber issues, we did not see the return in doing so and instead focused on functionality.

4 Implementation

Like our design, our implementation consisted of two major components the android application and the computer vision pipeline. Most of the learning and challenges associated with our project came from the computer vision pipeline, so we will discuss that implementation first before discussing the android implementation and finally the plugin we used to combine the two. Additional information related to the implementation of the project can be found in our detailed development guide in Appendix III. Next implantation steps based on our final implementation can be found in Appendix II and restated in Appendix III.

4.1 Computer Vision

Starting from last semester, we made a major pivot in the computer vision strategy we were employing because we hit a wall with what we were being able to implement. This stemmed from the attempted use of partially working solution attempts online, but they were too complex to change to get working well. So, we decided to take a step back and simplify the problem we were looking to solve, thus the addition of colored dots onto the chessboard so allow for color finding vs. arbitrarily finding things like board corners. In this new iteration, the general process is outlined as follows: read in the image to scan, find the board corners, shift the perspective of the image, so only the board is visible, determine which spots are open or occupied, and finally figure out the move made between pictures.

Kicking off the new computer vision pipeline, the reading of the image to scan mostly stayed the same from our previous iteration. The main difference is that we now provide the options to read from a specified file or to choose from a popup window, making testing different images on a computer much easier. We also change the color space of the image to HSV to allow for the specification of color masks ranges to be done with differing hues instead of trying to figure out the RGB ranges.

The steps from here on are where we changed from our previous iteration into this new one. The first of these new steps, the finding of the board corners, starts with creating a mask of the color green using a specified HSV color range. From here, the mask is blurred to get rid of small green pixels and clean up the mask. With the blurred mask, the bulk of finding the corners is done by searching and finding circles using OpenCV's Hough Circles method. With circles detected, we check to make sure we only found four, sort them in clockwise order, and make sure they form a mostly rectangular shape. If these properties are met, the method returns the corners found for the perspective-shifting of the image, if they are not met, the method changes the range of green that is used to create the mask, and the whole cycle is repeated a set number of times until four corners are found. Eventually, if 4 corners are not found, then the vision pipeline errors out and waits for another image to try again. An image of a successful corner find is found below in Figure 4.

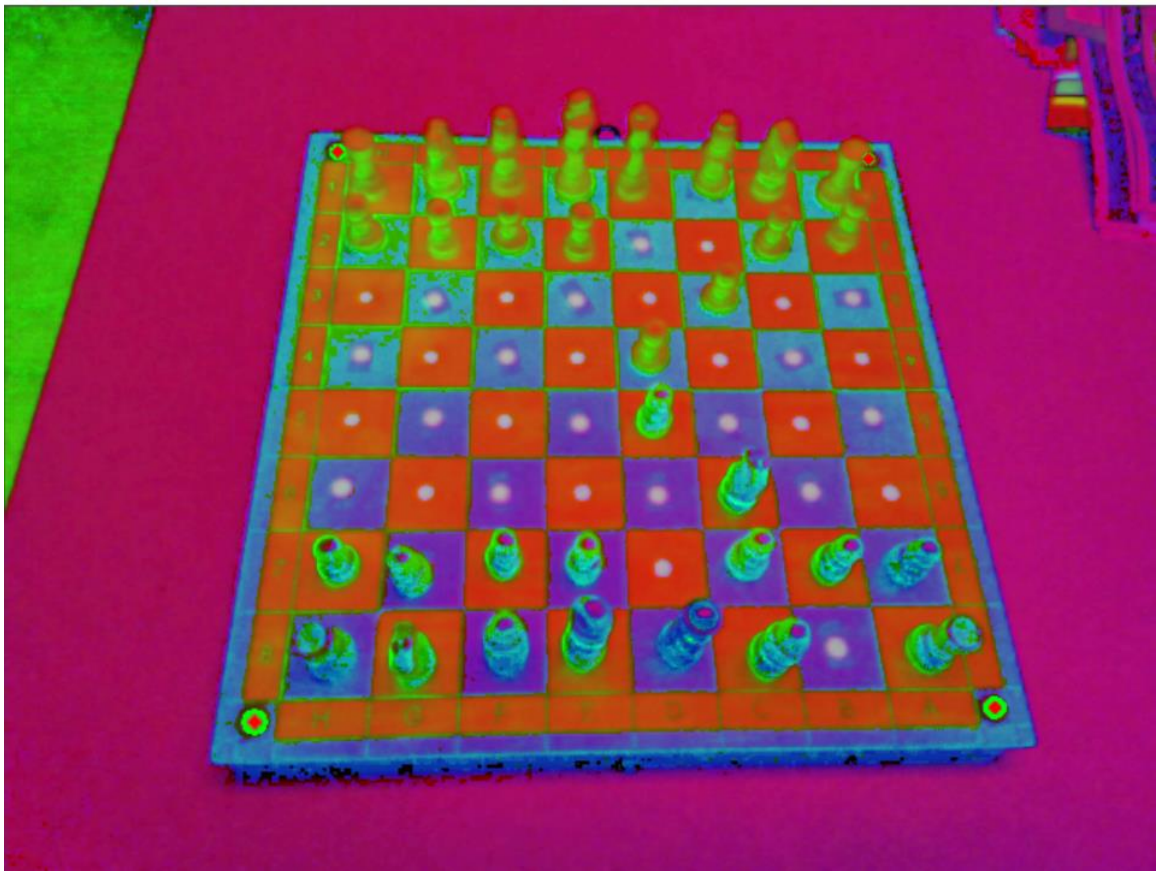


Figure 4: Successful Corner Find

With the four corners successfully found, the board image can be shifted such that it appears as a top-down image. Achieving this perspective shift is straightforward, starting with figuring out the size of the perspective image we will generate. The size is found using the smaller of the height and width so that we shrink instead of stretching the image. Next, we set up the image points and how they will map between the source image and our output image, in this case, the source points are the corners, and the destination points are the corners of the edge-sized image. Finally, we calculate the perspective matrix using OpenCV's getPerspectiveTransform method and then warp the source image using warpPerspective. An example of this warped perspective image can be found below in the figure *Warped Perspective*.

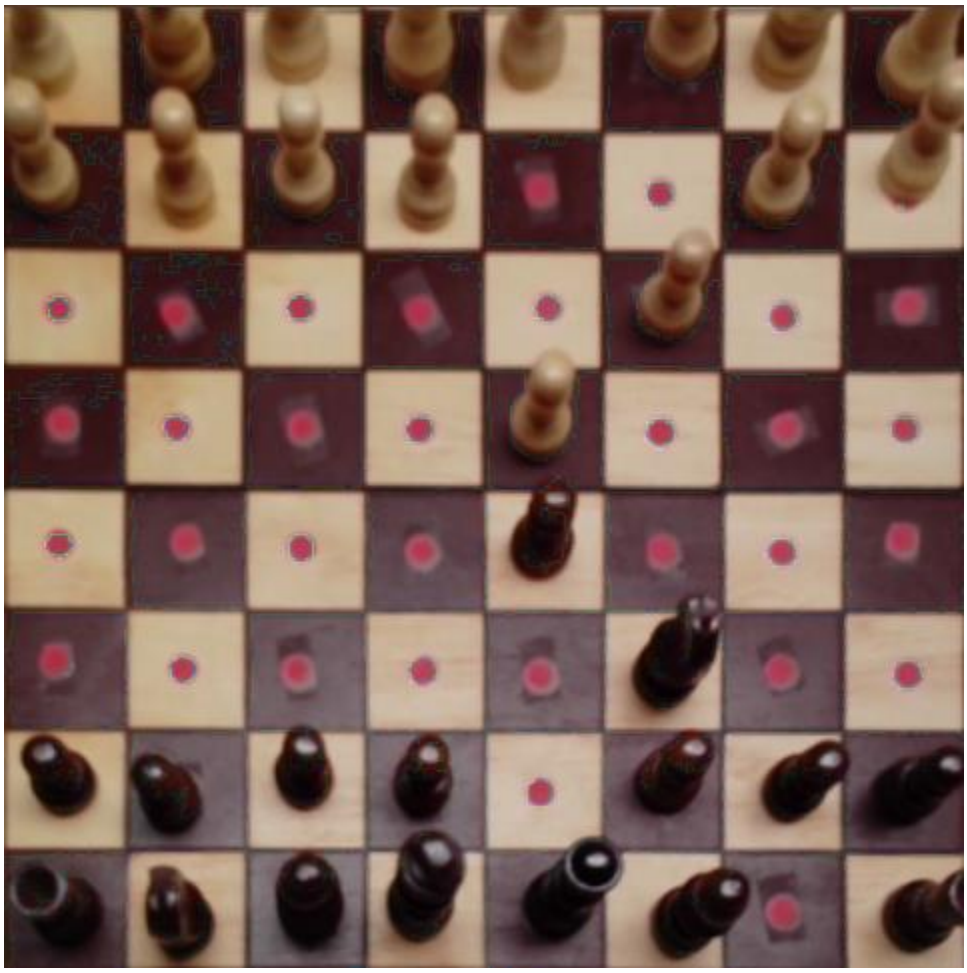


Figure 5: Warped Perspective

Now we are onto what is the most complex and open part of the computer vision pipeline, figuring out the board state. This part of the pipeline is like the finding of the green corners, with the primary differences being that we check cells individually and we look for piece color along with the red circle. To start finding the cell states, we segment the board by taking the edge size previously found and splitting the image into 64 squares since we have an eight-by-eight board. Next, we iterate over each individual cell and check if it has a red circle using the Hough Circles method, and if not, what color piece occupies the space. These are done in similar ways to the green corner finding as predetermined color ranges determine the found. Once all the cells are iterated over, we end up with a 2d array representing the state of the board based on the image scanned in.

The final major step of the computer vision pipeline is figuring out the move made between the current and earlier image when running the looping version of the computer vision pipeline (just adds in the keeping track of board states and figuring out of moves). In this step, we take the newly found board state and make a comparison of each cell to the previous board state to generate a new board array representing pieces that got moved. With this movement array, we can determine simple moves made if only two changes are found, matching our use case restrictions. If more than two changes are found, we know that the scan did not work as well as expected and we throw out

the newly scanned board and wait for another image to try again. Additional details on our implementation of the computer vision pipeline including functions and code documentation can be found in the corresponding computer vision sections of Appendix III.

4.2 Android Application

Our front-end development began in early October, and we were able to get a lot of satisfactory progress made last semester before going on winter break for December/January. We were able to hit the ground running this semester with a front-end that had most key screens completed as well as the communication, encoding/decoding, and pre-processing image libraries all ready to go.

For a clear idea of what the application screens look like, Figures 6 and 7 show the application home screen on both an emulator and through the glasses point of view. As shown, the Google Pixel emulator is not exactly to scale; however, the features are all still present, and most are functional for development purposes. The biggest constant throughout the application is the scroll bar at the bottom of the screen. This scroll bar is Vuzix's standard way of navigation within the device and was decided for the application to keep navigation consistent with what a Vuzix user would expect. Using the buttons and a touchpad on the side of the device, users can scroll through these menus and select the tile they want, for example, starting a game or taking a picture.

Figure 6: Home Screen on Pixel Emulator

Figure 7: Home Screen on Vuzix Blade

Below in Figure 8, you see the output of the Image Capture Activity when running the application on the Vuzix Blade device. Note that this Image Capture Activity is unable to be recreated on the Pixel emulator like the screens above because it requires camera access that the emulators do not provide. However, this screen is quite simple, it has two buttons, a back button and a take image button in the bottom scroll bar, and the rest of the screen shows you a camera preview so that you can see what the picture would be off before you capture the image.

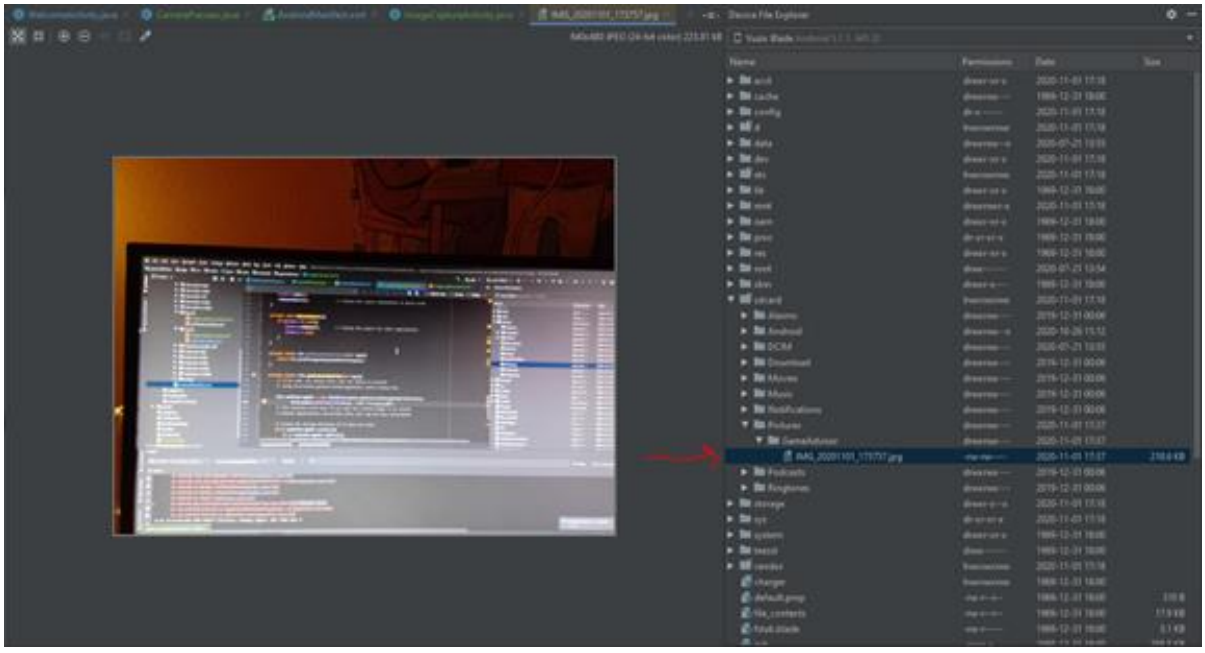


Figure 8: Proof of Image Capture

While the computer vision required a large pivot at semester, the only change we had to make on the front end was to run Stockfish (the applications chess engine) in the Vuzix Blade's Android front end code instead of the Python backend. This change was made due to difficulties encountered when running the Python backend on the Vuzix Blade itself. Our team was prepared for this possibility, however, and had already developed a class in Android to handle Stockfish

during the Fall semester. This allowed the change to be implemented quickly and without very much trouble after the decision was made.

To play the game, a user will select to start a game and be taken to the gameplay screen, as seen in Figure 9. From there, our gameplay loop starts, and the user will take a picture of the game board, which is saved to a specific location on the Vuzix Blade. The Computer Vision back end then compares the board state of the image taken with the game's previous board state and saves to a file the move detected or an error message. The front end reads this file, and in the event of an error, the user is prompted with what must be done; for example, try a different angle and re-take the image. If the file contains a move, the front-end sends it through our Stockfish implementation and toasts the user with the recommended move for them to make, shown in figure 18. This gameplay loop continues until gameplay is finished.

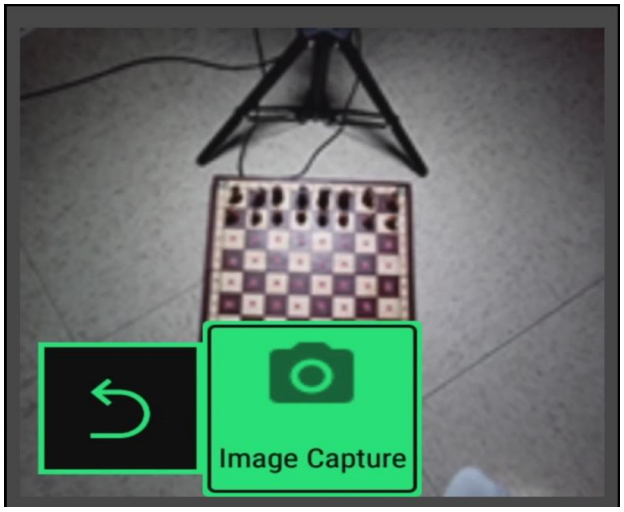


Figure 9: Gameplay Screen as Seen through the Vuzix Blade

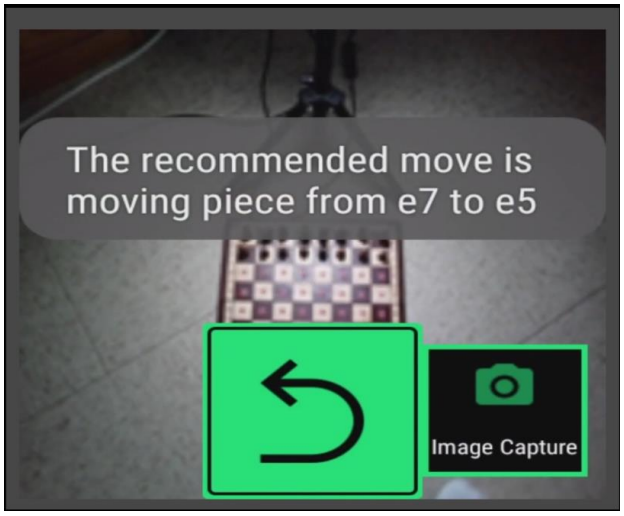


Figure 10: Recommended Move displayed on the Vuzix Blade

4.3 Chaquopy Plugin

The Android and Computer Vision components discussed in the previous two sections were implemented separately with the goal to integrate the computer vision pipeline into the android application. However, the computer vision module was written in Python, while Android uses Java or Kotlin. This required us to find a library that would allow us to run the python code directly on our game advisor application. We decided to use Chaqupoy to allow us to execute the computer vision pipeline within the Gameplay Activity, which is written in Java.

Chaquopy is a grade plugin for Android that allows us to execute python code within the application. It can be added to any android project by adding a few lines of configuration in the top-level and module-level build.gradle files. Once added, your application can successfully run python code as if it were natively supported. The only requirements from Chaquopy are minimum sdk and gradle versions, both of which go back extremely far if you use an older version of the plugin, and therefore unless your application is supporting extremely old versions of Android the plugin should work for you.

While not a direct requirement, if you use the Chaquopy library, you will likely need to either buy a license or make your project open source to get a free license otherwise, your app will shut down after five minutes. Since our application is still an early prototype, we did not purchase a license or make our project open source. However, if we were to release our application, we would likely make the project open source to get a free license rather than purchase one.

5 Testing

Testing and test results are a great indicator of the success and progress of our project. Throughout the project, we have a variety of testing methods, including unit, interface, and user testing. Each of these is discussed below in detail.

Additionally, throughout the project, we performed regression testing. Regression testing is especially important to the project's computer vision aspect as we incrementally tweaked algorithms to determine the best performance. For front-end and backend modules, these regression tests were less frequent and coincided at a minimum for each of the major releases (Prototype, Minimum Viable Product, and Final Design).

When developing our tests and overall testing plan, our Test Lead Brett, and others involved, consulted the Software Quality Assurance Process Standard [4].

5.1 Unit Testing

For the Android application, we had truly little unit testing as most of the screens were verified as part of our interface testing. The only unit testing on the front end revolved around the Stockfish engine and our library for communicating with it. This testing was minimal when compared to the computer vision and focused on sending commands and decoding and verifying the output from the engine. Therefore, most of our testing was related to the computer vision module.

In order to test our computer vision implementation, we have implemented regression testing on a series of test images. The test images were taken via the Vuzix Blade device or a phone camera and contain a variety of known game board states and orientations. These testing results were compared with the known board states and allowed us to incrementally develop the computer vision module. The computer vision implementation unit testing is fully automated.

5.1.1 CV Module Unit Testing Plan

To test the computer vision module, we are using unit tests with a test suite containing many pictures of full chess games as taken through the glasses and other sources. In each unit test, we feed testing images to the CV module in sequence and compare the result with the known moves for that chess game using PGN notation. We established a testing suite with many full chess games and their corresponding images taken through the glasses. This uses Python's unit test framework.

Not all of these unit tests are passable by our version of the app, such as tests that include promotion and en passant captures. However, we included these tests in the testing suite so that the project can be expanded in the future to include some of these edge cases.

Different test cases each followed a distinct set of conditions on which the functionality of the CV module depended on:

- Different test games are taken in a variety of areas to account for changes in lighting conditions or backgrounds.
- Each test game has some variation in the top-down camera angle to account for different user's height and table height or shifting positions between moves.

- To fully test game state recognition, test games include a variety of moves such as captures, castling, en peasant, check/checkmate, etc.

The following is a list of test games that were used and a brief description:

Game 1: The first test game has no captures and simply basic piece movement.

Game 2: This test game contains a few simple captures.

Game 3: Longer game with more captures, this time including back-to-back captures and checks.

Game 4: Captures, checks, castling on both sides, promotion to the queen. (Ends with resignation)

Game 5: Game including en passant capture.

Game 6: Famous Opera game

Additionally, some test games are repeated with the addition of stalling between moves. Multiple of the same image are passed to the CV module to simulate no move being made.

Game 7: Repeat of game 1, with stalling between moves.

Game 8: Repeat of game 4, with stalling between moves.

Game 9: Repeat of game 6, with stalling between moves.

Each of these test games is repeated under two different lighting conditions or backgrounds.

Furthermore, in addition to full game tests, we also have individual unit tests for some of the functions that are used within the computer vision module. This includes tests for find_board_corners, get_board_diffs, and get_last_move.

5.1.2 CV Unit Test Results

The unit tests written for the computer vision module were run many times throughout 492 to verify the progress of the project. In this section we have screenshots of the tests that were discussed in the previous subsection:

```
PS C:\Users\txcyu\Desktop\CV\sdmay21-05\ComputerVision> python test_CV.py -v
test_getboarddiffs1 (__main__.TestCV) ... ok
test_getboarddiffs2 (__main__.TestCV) ... ok
test_getboarddiffs3 (__main__.TestCV) ... ok
test_getboarddiffs4 (__main__.TestCV) ... ok
test_getboarddiffs5 (__main__.TestCV) ... ok
test_getlastmove1 (__main__.TestCV) ... ok
test_getlastmove2 (__main__.TestCV) ... ok
test_getlastmove3 (__main__.TestCV) ... ok
test_getlastmove4 (__main__.TestCV) ... ok
test_getlastmove5 (__main__.TestCV) ... ok

----------------------------------------------------------------------
Ran 10 tests in 0.003s

OK
```

Figure 11: Unit Test Output for get_board_diffs and get_last_move

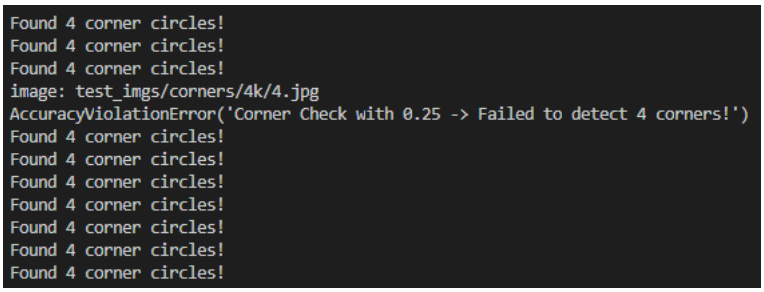


Figure 12: Output from running tests on find_board_corners on a set of 4000 Images

The failure seen in the image above is due to test image having a green object in the background.

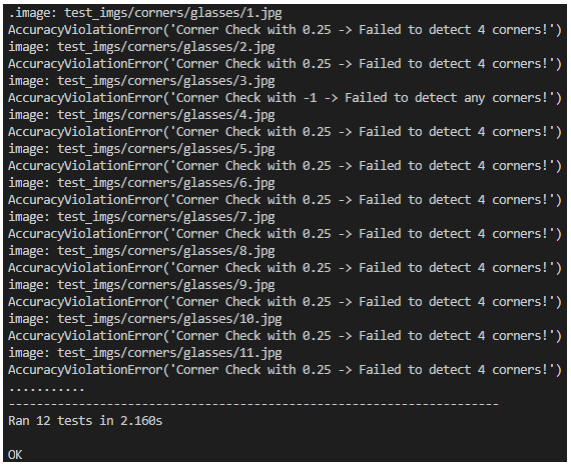


Figure 13: Output from Running Tests for find_baord_corners on a set of uncalibrated images taken by the glasses

To improve the accuracy of detection, we added calibration functionality to the application that modified the color values we were looking for based on the current environment—doing so significantly increased detection accuracy, as before calibration, we struggled to detect the state accurately as seen in the image above.

5.2 Interface Testing

In our project, the interfaces correspond primarily to user interfaces on the glasses. We have a variety of screens that use the communication and/or game libraries that were manually tested. These interfaces were tested both on Android Emulators and on the actual Vuzix Blade device. All these interfaces have been verified by team members as acceptable in terms of functionality as well as ease of use. These interfaces, as well as an explanation of them, can be found below:

- Welcome Activity: This activity is shown when the user powers on the glasses. It acts as a welcome screen with menu options.
- Image Capture: This activity is used to capture images to be used for testing. The camera preview class used in this activity is also used in the main gameplay activity.
- Gameplay Activity: This acts as the main activity for the app, which has a camera preview and shows the user the recommended move.

- Settings/Difficulty: This activity allows the user to customize the game engine's difficulty and in the future timing, such that if they are playing under time constraints, the engine gives the user a recommended move within the user's requirements.

In addition to our regression testing (described above in section 5.1), we have interface testing for the communication between the Vuzix blade app and the CV module. These interfaces include:

- Storing the image: This involves saving the image taken through the Vuzix blade to a designated location on the device, which will then be read from by the CV module.
- Storing the recommended move: This involves storing the recommended move by the CV module to a file on the Vuzix Blade, which can then be read by the android app and displayed to the user.

These interfaces have been manually tested and verified, as they involve writing and reading from files. As these interfaces were tested individually and by hand, we simply used mock data and test images initially, before switching to real data when things were integrated.

5.3 Acceptance Testing

Originally, our plan for acceptance testing involved two levels. The first level of testing involves testing the combining of multiple components in the app (described below). The second level of testing was going to involve integrating all the components and playing full games. In practice, most of our time has been spent on the first level of testing, since the different components of the project all presented challenges which made smooth full-game tests infeasible.

The first level of our acceptance testing involves combining two or more interfaces together to create a more complete flow through the app. These flows and their description can be found below:

- Welcome and Gameplay: The user turns on the glasses and is presented with a welcome screen. Upon selecting gameplay, the user can start the game with a physical chessboard in the starting position and be shown recommended moves with each image sent to the CV module.
- Image Capture with Calibration: The user can capture images to be sent to the computer vision module. As this is done, the computer vision module will go through calibration steps to optimize the algorithm for lighting conditions.
- Game Engine Customization and Displaying Result: The user can customize the engine through the app to meet their desired difficulty setting and can see more precise move recommendations as a result.
- Computer Vision Pipeline: The project should be able to correctly identify board states of test images with known correct values with a certain level of accuracy. Sample images are provided with the project, and a testing framework is provided for the user to do their own tests.

These level one test cases are used to test each of the individual functional requirements (1.4) amongst the development team. These level one tests have been tested individually and by hand, using mock data when necessary, throughout development and real data now that integration is

complete. Along with the creation of our testing plan, we have determined the following acceptance criteria by which to judge our level one tests:

- Capture the board using the AR glasses
    - During a game, when an image is captured, the image shall be saved by the android app and then read by the computer vision module to detect and process the board state.
- Detect and process the board state using computer vision (CV) algorithms
    - Upon receiving an image, the CV module will perform any necessary calibration steps to account for lighting conditions.
    - Upon having a valid image, the CV module will calculate the empty and occupied squares, such that it is known whether each square of the board is occupied by a white or black piece. (Not detecting individual pieces)
    - After detecting empty and occupied squares, the CV module will calculate the actual board state based on the differences from the previous board state.
- Determine the best move given the current state using top game engine(s)
    - After the CV module calculates the board state, the board state is sent to Stockfish with the correct difficulty settings and calculates the recommended move.
- Indicate recommended moves to the user on the AR glass display
    - After receiving a result from Stockfish, the recommended move shall be displayed to the user on the AR glass display.

We followed these criteria to judge our confidence about the passing of our level one tests. The goal is to integrate all the components into a final design and show a working pipeline.

The second level of acceptance testing focused on real full game user testing with Dr. Zambreno and those passionate about chess and interested in the device. Since we never completed the fully viable product, we never got around to this level of acceptance testing. However, we had plans to meet with the ISU chess club as well as Dr. Zambreno multiple times to allow them to use the device and provide feedback to us, which we would iterate on in the next sprint. Unfortunately, we were not able to perform these tests this semester.

5.4 Results

As mentioned in the sections above, much of our testing, especially on the front end, has consisted of eye tests. The exception was the automated unit tests for the computer vision pipeline, which is discussed in its own section above and will not be restated here. Both the eye tests and a significant majority of our computer vision tests were successful, with some cases of inaccuracies persisting and certain edge cases not being accounted for.

On the backend, our results look different from the end of last year because of our pivot in computer vision strategy. We included the ability to print out many of the various stages of the pipeline for debugging purposes, such as the different masks we create, highlights of the colors found, and the final warped perspective of the board. When it comes to testing the computer vision beyond the unit cases, most testing is done by hand for quick iteration as minor changes are being made, but we have some tests set up that would be used once the computer vision is more consistent.

Below, in Figures 14 – 18, we can see the output of our computer vision pipeline at different stages. Each of these stages was eye tested throughout development and as tweaks were made to see how those changes affected the pipeline. The goal of printing these images was to allow us to easily try and fine-tune things and see how our results were affected as well as to debug the pipeline.

Figure 14: Original Image

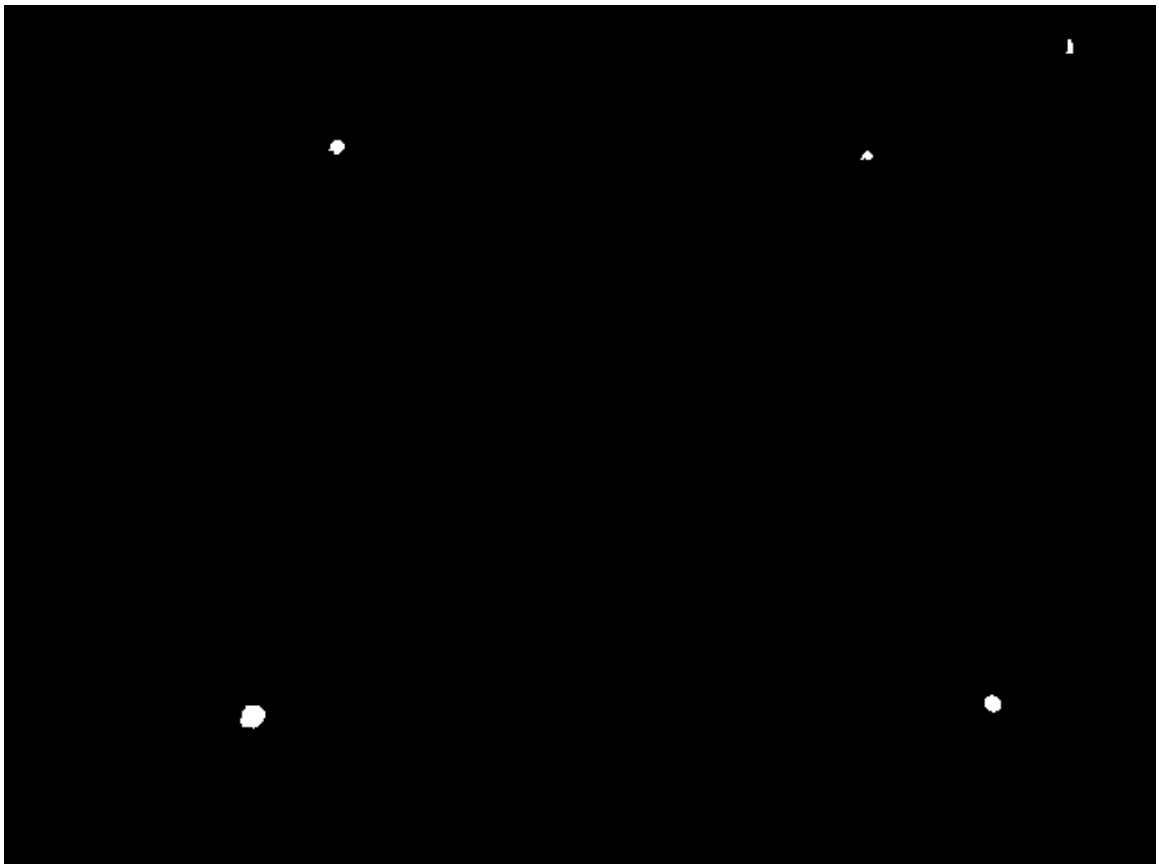


Figure 15: Mask of Green Pixels

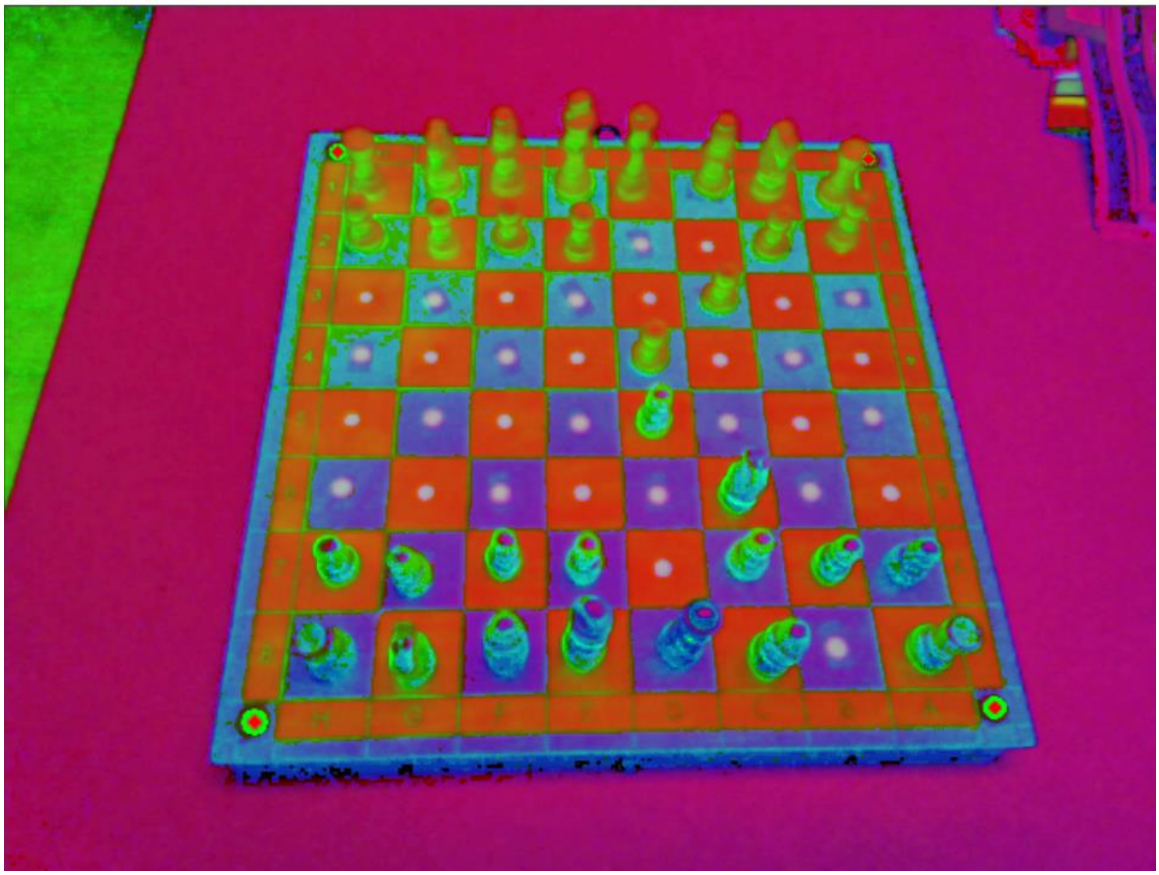


Figure 16: Highlighted Corners

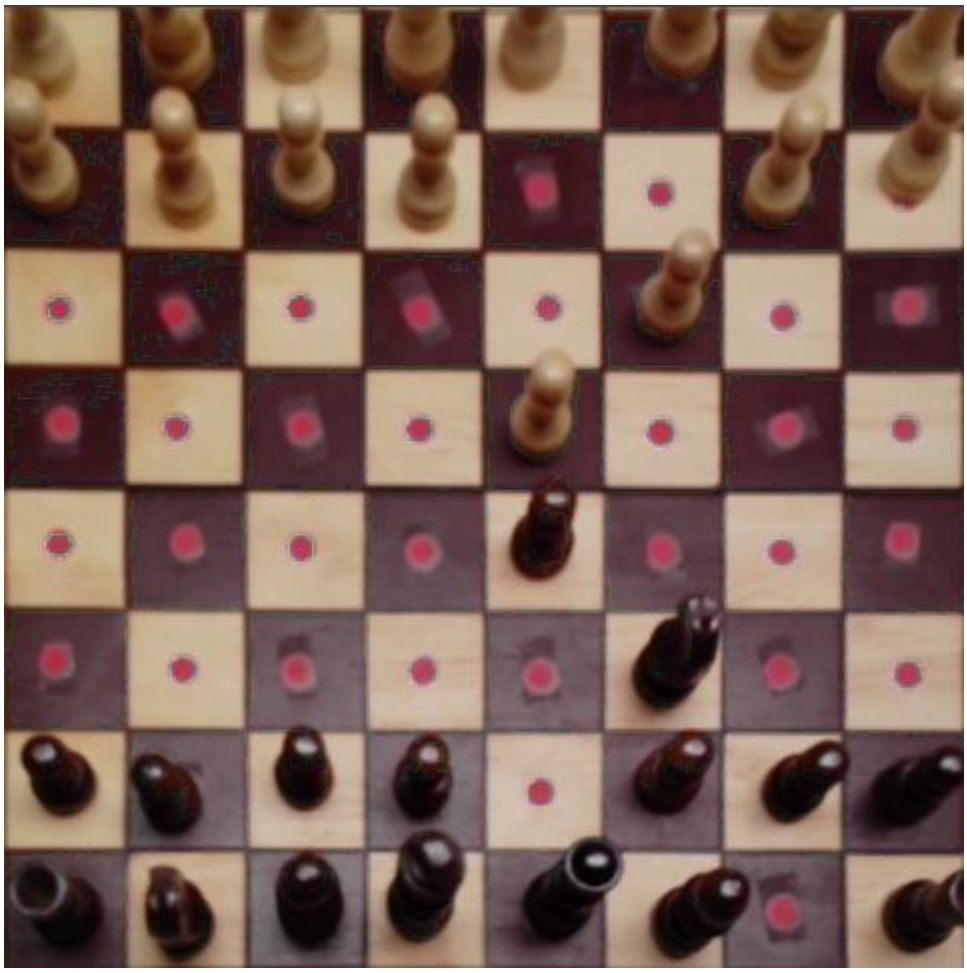


Figure 17: Warped Image Perspective

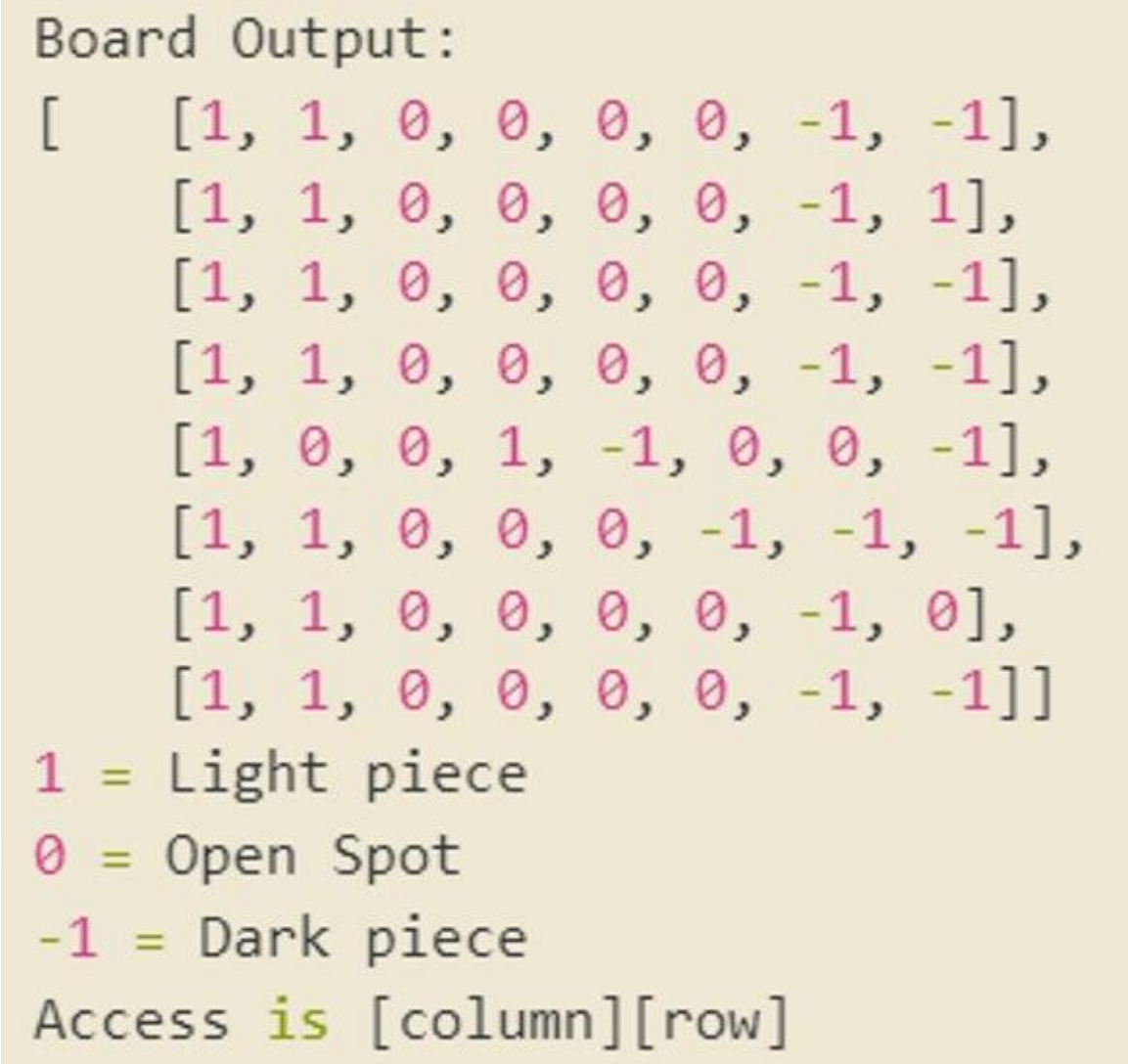


Figure 18: Board Representation

These images help show the debugging integrations made in the computer vision pipeline notebook and allow us to visually inspect and test the results of the pipeline at different stages. These outputs have been critical in both our development and testing.

6 Closing Material

6.1 Conclusion

In CPRE 491, we successfully completed market research in the areas of AR Glasses, Computer Vision, and Chess Engines and selected the devices and libraries that are the best fit given the project requirements defined in section 1.4. After selecting the components for our project, the team began working on planning out the project to achieve the goal of developing an application that allows a user to play chess with the help of or against our selected chess engine.

Our project plan focused on three main components: AR Glasses, Computer Vision, and Game Engine. This division was mainly driven by the requirements of the project as well as the natural modularity of these areas. Within each area, we have a variety of tasks and submodules that keep our project as modular as possible. We also planned for additional communication submodules in each of these three modules in case the scenario appears where the backend (computer vision and game engine) must live on a separate host such as a mobile companion app. By planning out the project this way and breaking up the tasks as we did, we left ourselves with great flexibility to pivot to another strategy board game or module location if necessary. Our project plan is very detailed, and for further details, please visit sections 2 and 3.

In 492, we restarted our implementation of the computer vision pipeline as needs arose for reduced complexity. While this implementation was nearly a full restart, the new pipeline was caught up with our 491 pipeline in just a few weeks. The work on the Android Application continued as planned, and only small implementation tweaks were needed throughout this development. As the semester progressed, we began to develop in a much more iterative style than we used in CPRE 491. This allowed us to try and fix pain points and make improvements as we prepared for the Industry Review Panel Demo. Additionally, these iterations allowed us to fix problems uncovered from testing the system in the past bi-weekly sprint. By the end of the semester, we had a functional demo program that allows for semi-consistent gameplay for five minutes, at which point the app will close due to licensing. There is still substantial work that can be done to improve the board detection, despite our best calibration efforts this semester. As a result, the device is still far away from a production-level product. However, we feel we made substantial progress and have a well-written and documented project that not only is Dr. Zambreno happy with but can be passed off to the next team or independent study student.

6.2 References

[1] Osterlund, P. (2020) Droidfish (Version 1.84) [Source code] https://github.com/peterosterlund2/droidfish

[2] Sam (2018) ChessboardDetect [Source code] https://github.com/Elucidation/ChessboardDetect

[3] C. Danner and M. Kafafy, "Visual Chess Recognition," Stanford University, Stanford, California,

[4] "IEEE Standard for Software Quality Assurance Processes," in IEEE Std 730-2014 (Revision of IEEE Std 730-2002), vol., no., pp.1-138, 13 June 2014, doi: 10.1109/IEEESTD.2014.6835311.

[5] "IEEE Standard for Software Reviews and Audits," in IEEE Std 1028-2008, vol., no., pp.1-53, 15 Aug. 2008, doi: 10.1109/IEEESTD.2008.4601584.

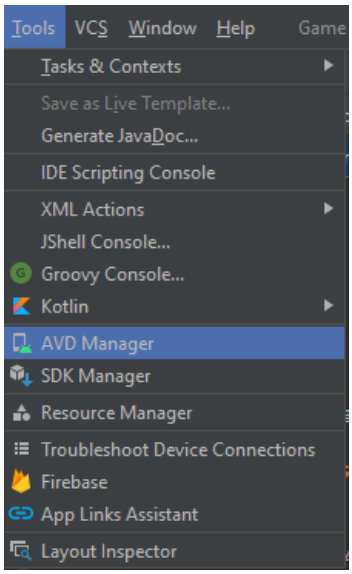
[6] "ISO/IEC/IEEE International Standard - Systems and software engineering — Developing information for users in an agile environment," in ISO/IEC/IEEE 26515:2018(E), vol., no., pp.1-32, 20 Dec. 2018, doi: 10.1109/IEEESTD.2018.8584455.

[7] C. D. Farinango, J. S. Benavides and D. M. Lopez, "OpenUP/MMU-ISO 9241-210. Process for the Human Centered Development of Software Solutions," in *IEEE Latin America Transactions*, vol. 13, no. 11, pp. 3668-3675, Nov. 2015, doi: 10.1109/TLA.2015.7387947.

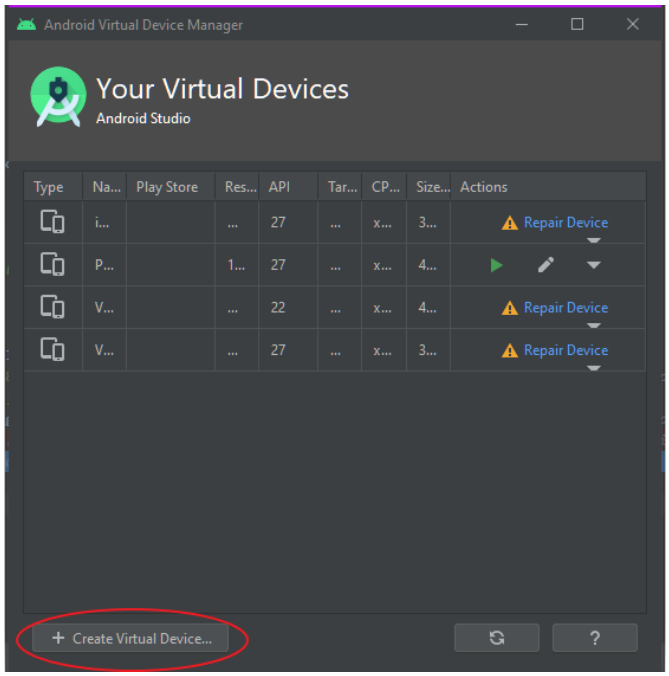
Appendix I: Operations Manual

Vuzix Blade Emulator Setup in Android Studio – Only necessary if you Are planning to Develop for the Glasses without having a physical Device

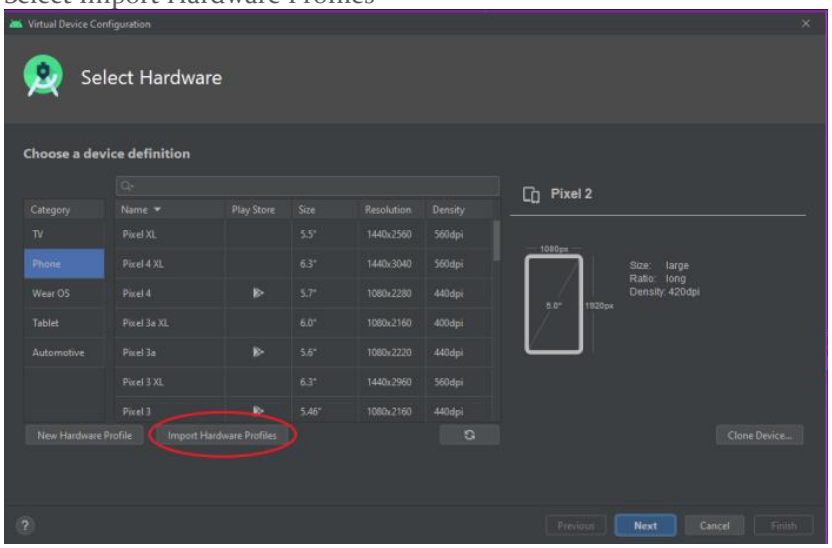
1. In Android Studio Go to Tools -> AVD Manager



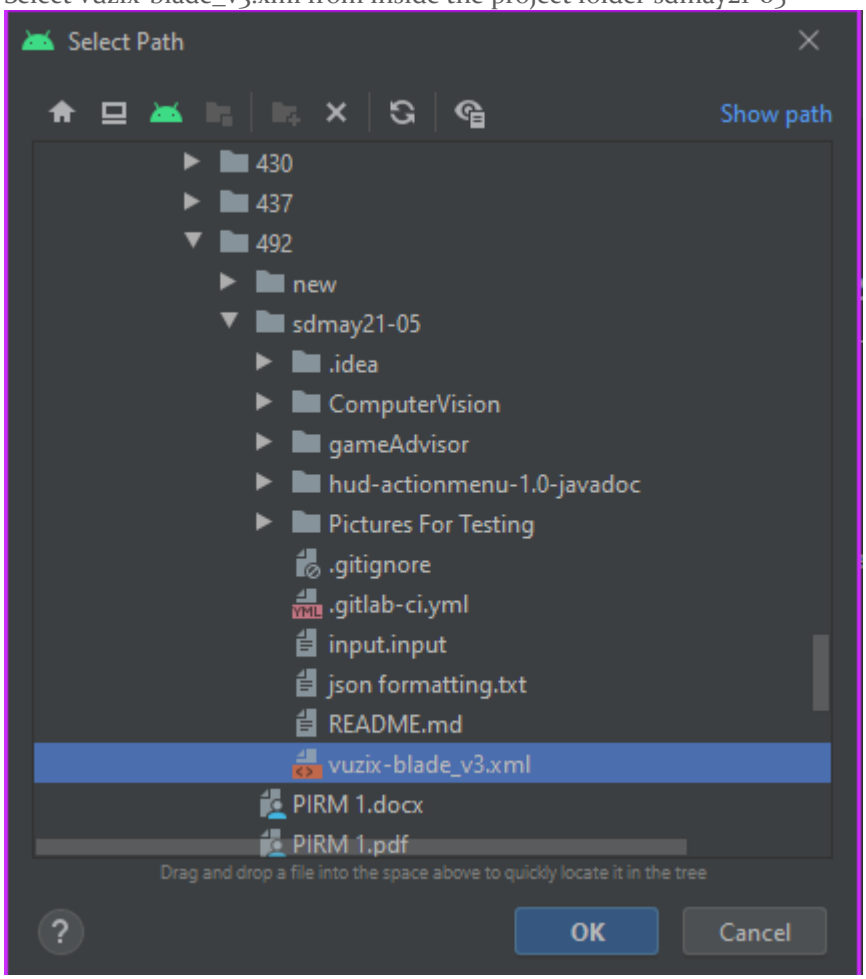
2. Select Create Virtual Device



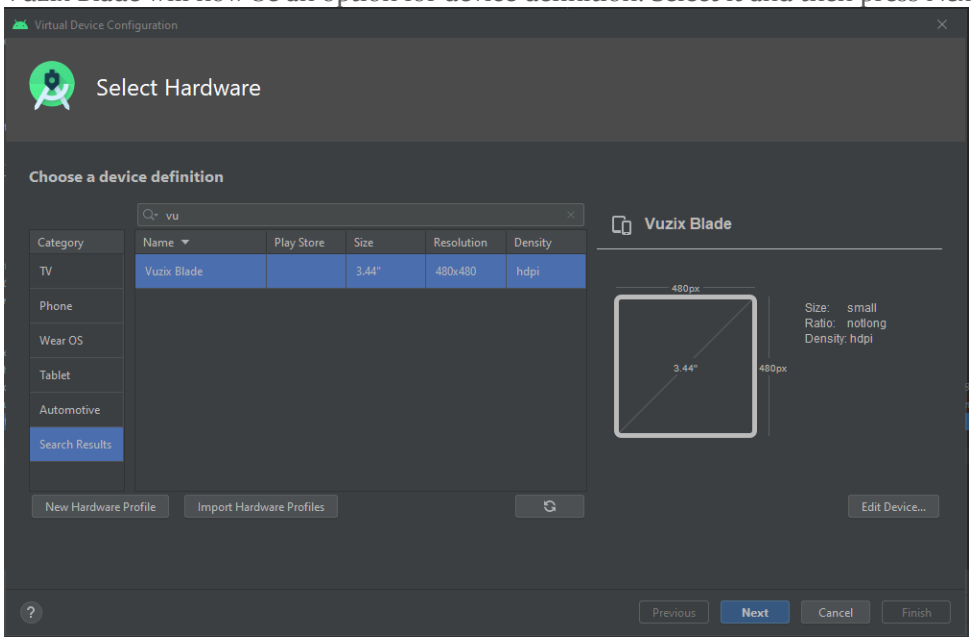
3. Select Import Hardware Profiles



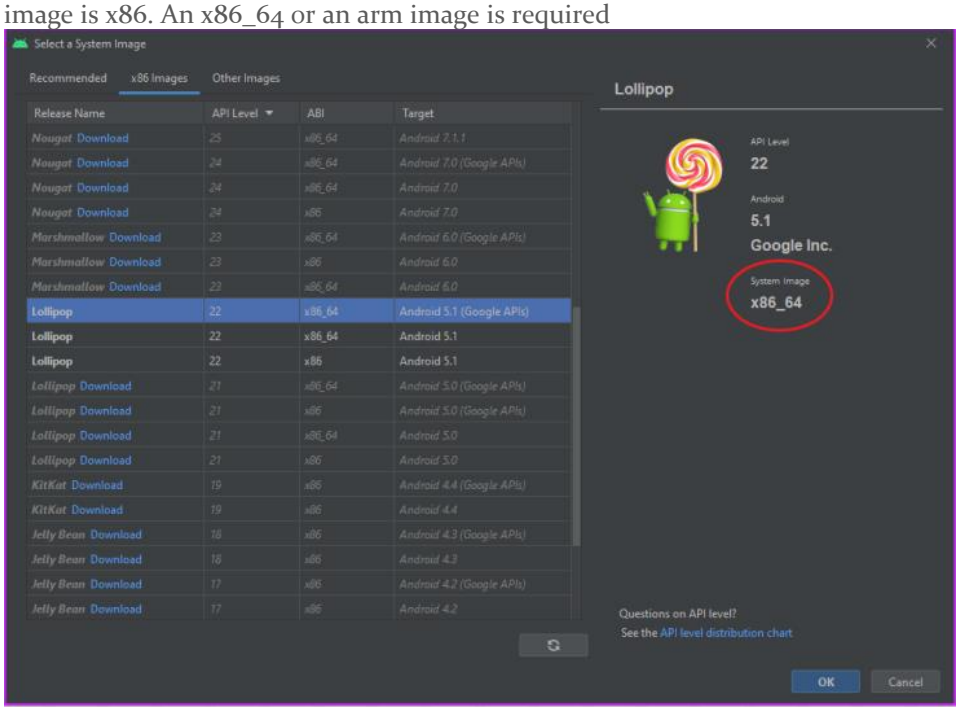
4. Select vuzix-blade_v3.xml from inside the project folder sdmay21-05



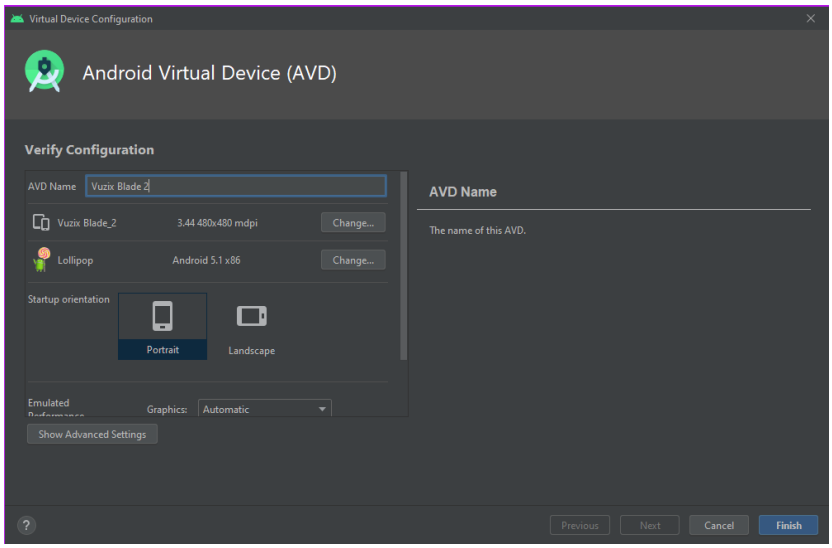
5. Vuzix Blade will now be an option for device definition. Select it and then press Next



6. Select system image: Lollipop/API 22. System image will not work with Stockfish if the image is x86. An x86_64 or an arm image is required

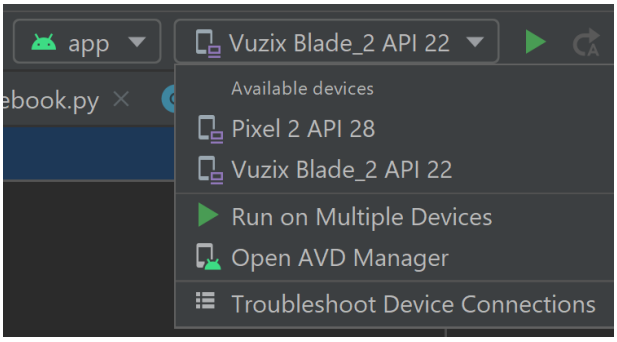


7. Check information is correct and select Finish



VUZIX BLADE SETUP AND DEPLOYING TO THE BLADE

1. Power on the Vuzix Blade
2. Navigate to Settings -> Developer Options.
3. Enable File Transfer and Developer Modes
    a. You may also want to turn on Android Debug Bridge (ADB) however this is not required.
4. Connect the Vuzix Blade to your computer using a Micro-USB cable such as the one provided in the box
    a. Note: The cable in the box is a bit of a pain to work with, and we recommend users get a longer cable that they know works
5. The device should now be selectable from the list of available devices as shown below:



    a. Occasionally the device may not appear here. When this happens, we recommend users click “Troubleshoot Device Connections” in the above photo and follow the recommended steps.
6. Once you have selected the device, you can now press the Green Play Button next to the device name (see picture above) to deploy the application to the Vuzix Blade

a. Alternatively, users may deploy the app using ADB Wireless Debugging. However, we did not do this and recommend users physically connect the Vuzix Blade to the computer and deploy using the Micro-USB. For those that do need or want to use ADB Wireless Debugging, please consult the following: https://developer.android.com/studio/command-line/adb

Using the Game Advisor Application

1. If you have just deployed the Game Advisor Application from Android Studio, the application should open automatically. However, for those that are not installing the app for the first time, you should scroll through the home page to find the Game Advisor Application and select it by tapping the touchpad on the right side of the glasses as if you were tapping your temple.
2. Once the application is launched, users should select "Start Game" to begin playing.
3. Once the user sees a camera preview, the glasses are ready to accept photos of the chessboard. To capture photos, please scroll to "Capture Image" using the touchpad located on the wearers right temple on the glasses and tap this touchpad to select this button and capture the image.
4. The Game Advisor Application will now process the image and return a message to the user indicating the recommended move, at which point the user should execute it. In some scenarios, the image captured may be poor, and therefore we cannot accurately determine the board state. In this case, the user will be prompted with an error message and should take another picture.
5. Steps 3 and 4 should be repeated until the game is completed.

Testing the Computer Vision Pipeline

1. Open jupyter labs by running "jupyter lab" from your terminal.
   a. If you have not isntalled jupyter lab please visit the Jupyter Lab Section in Appendix III for instructions on how to do so.
   b. Note you may need to use the full path if jupyter labs was not added to your path variable after it was installed.
2. Navigate to the Computer Vision Folder in the projects Git Repository.
3. Select the Notebook you wish to run. The testing notebook is used to test certain functionality while the Complete Loop Notebook has very detailed and comprehensive printouts that are great for debugging and testing the entire pipeline.
   a. If you wish to run the full loop notebook in debug/test mode, you may need to change the configuration file to allow printouts at different stages of the pipeline.
4. Verify the pipeline passes the tests by visual inspection or through reports where applicable.
5. Run test_CV.py in this Computer Vision Folder from command line to test the functions not revolving around images (getboarddiffs & getlastmove)

Appendix II: Other Considerations

What We Learned – Computer Vision Pivot

As discussed in the implementation section for computer vision (Section 4) and in Section 3.8, we did end up making a fresh start of the computer vision pipeline using what we had learned from our first attempt. This was done because we ambitiously hoped we would have a solid enough understanding of the programs we were building off such that we would be able to improve it and add all the steps necessary to determine the board state. Unfortunately, as we started having to improve more complex parts of the pipeline, we found we were having difficulty making much progress, having trouble even getting an empty board to be recognized reliably. Noticing we were having this difficulty, we took some time over winter break to talk to some people and do more research online about best computer vision practices and eventually concluded that we should simplify the problem we are trying to solve by adding the colored dots to the board. This redesign also allowed us to improve the modularity of the program, with each step of the pipeline being able to be changed independently if the overall state is the same between different versions of individual steps. For example, in the future, work can be done on cell state detection that does not need the red dots to work. Once this new cell state detection is working, it can be directly integrated into the corresponding part of the pipeline if it returns a board state array.

Next Steps – Accuracy Improvement and Reducing Assumptions

We were able to develop a computer visions pipeline that can semi accurately create a board state from images. In doing so, we had to make a variety of simplifications and assumptions, including a known initial state, adding red dots to each board square, adding green dots to the board corners, and requiring traditional board sets. These simplifications and assumptions made the project much easier for us to implement as a proof of concept; however, they do not allow for full practical usage. For the next senior design or independent study group that takes over the project, there is clearly work to be done to remove some of these assumptions and simplifications. The first two things the next group or individual should work to remove are the red and green dots. These were added to greatly simplify detection capabilities by allowing us to determine space occupancy and the board's location based on these sharp contrast and consistent dots. However, since actual chess boards do not have these dots, detection capability needs to be modified and improved to not rely on them. One potential solution is to look in the center of each cell for a color that matches the color of a standard black/white piece rather than the red color of the dot. The detection and framing of the board are likely to be a more complicated problem, and we recommend that users investigate line detection as a potential way to solve this problem.
Once the dots are removed, ways to remove the remaining assumptions and simplifications may be explored. However, these assumptions will prove substantially more difficult. We restricted the gameplay of the application to start from the initial state so that we did not have to do piece identification, but if this assumption is removed, the pipeline will now need to not only detect if a space is occupied but by what piece. This will likely require training an ML model to recognize the pieces for each board cell that is occupied, at least on the first image captured. From there, the location of each piece would be known, and the current pipeline could be used to detect changes in the board state. Additionally, the computer vision could be expanded to include nontraditional chess sets such as Game of Thrones. However, doing so is likely overkill unless the product is trying to be released and sold publicly.

In addition to removing the assumptions and simplifications, there is additional work to be done to increase the accuracy of our board state determination. Currently, the detection only works in ideal scenarios with minimal shadows and glare. We started a calibration process to help combat some

of these issues; however, this process is not perfected completely and could still be fine-tuned to help deal with these scenarios. Additionally, there are scenarios where a piece is hidden from view, such as a queen block a pawn, that need to be handled. Currently, we require the wearer of the glasses to take another picture in this scenario, but logic could be implemented for this scenario to recognize this hidden piece in our board state. We recommend that the next group start making improvements to the pipeline as it relates to accuracy and consistency before moving on to removing assumptions and simplifications.

Next Steps – Expansion to Other Games

In addition to making improvements to the chess advisor, the next group of developers could move to implement other tabletop games that follow a similar flow. These games could include basic games such as Tic-Tac-Toe to more complex games such as connect-four and checkers depending on the developer's interests and skill levels. When designing and developing the game advisor application, we were as modular as we could be, allowing the next developers to reuse many of the existing classes, functions, and activities in non-chess implementations.

Appendix III: Developer Guide

Introduction

The purpose of this appendix is to explain the applications and codebase used for Game Advisor such that another team or student can pick up the project and continue development. For each section, we detail the information we feel a new teammate would need to know based on the time the document was written. It is possible that these instructions have changed since this document was written, in which case we recommend the user consult a combination of our information and their favorite search engine to find the answers.

Android Installation

Our project involves an Android Application and requires Android Studio. This can be installed by visiting https://developer.android.com/studio/index.html#downloads and downloading the correct file for your system. Once the file is downloaded, follow the installation procedure to install the application with the default settings. Note the latest version of Android studio should be compatible with our project; however, if it is not, we recommend that users use Android 4.0.X - 4.1.3 as these were the versions we used.

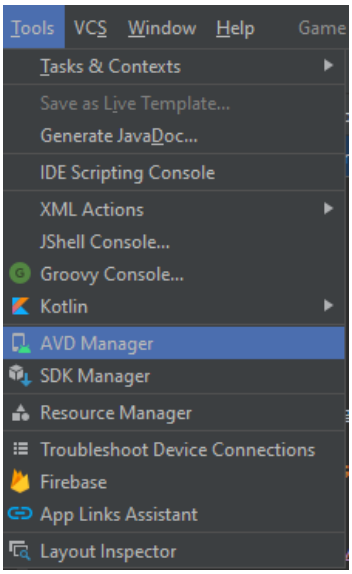
Android Settings

The Android Project uses the Chaquopy Library found here. This library is used to run the computer vision pipeline code within the application and requires certain project settings as detailed in the documentation. The current version of the library requires the project grade to be between 3.4 and 4.1 and the minimum SDK version to be at least 16. If, for some reason, you wish to support older Gradle versions, you can use an older version of the Chaquopy library. Older versions of the library are capable of supporting as far back as Gradle 2.2. No other special settings should be required to build and run the project. So as Android versions progress, natural changes to the remaining settings can be made.

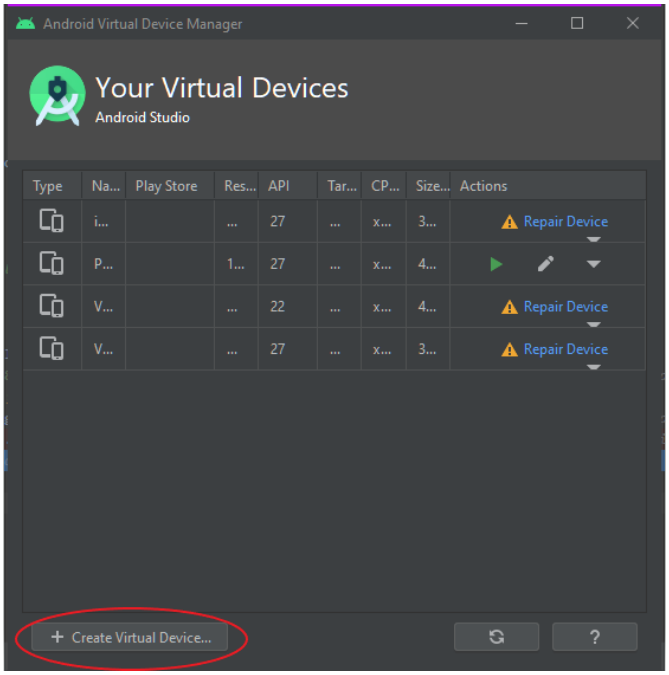
Android Emulator Setup

For those who do not have a physical device to build and deploy the Game Advisor Application to, we recommend you create a custom emulator using the instructions below. This emulator mimics the device architecture and software and should allow users to test some of the functionality. This functionality is going to be limited as there are components on the glasses themselves that are not mimicked well by an emulator; however, the emulator setup below gets developers close.

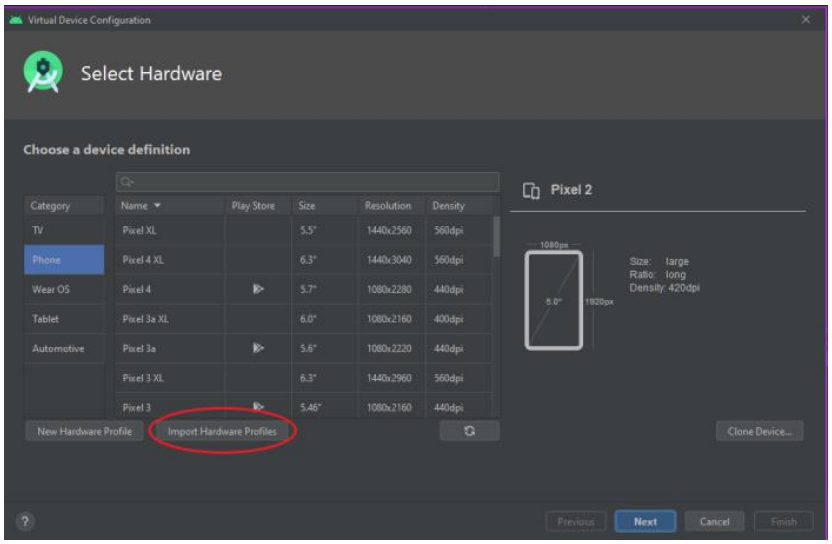
1. In Android Studio Go to Tools -> AVD Manager



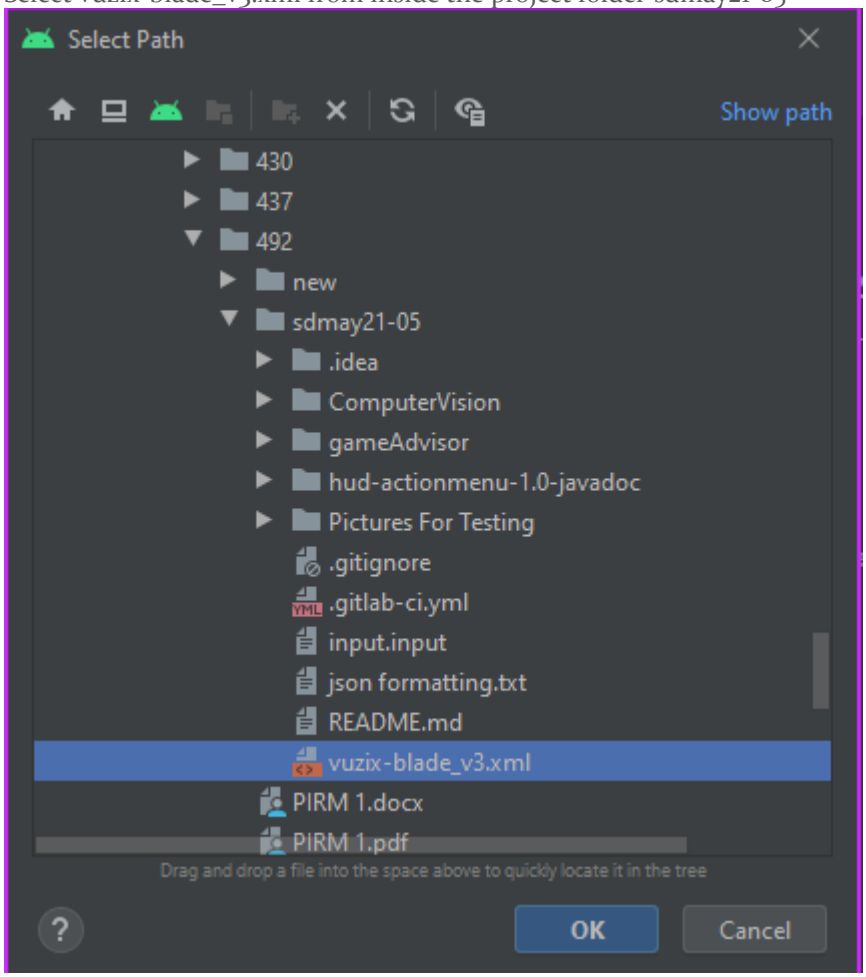
2. Select Create Virtual Device



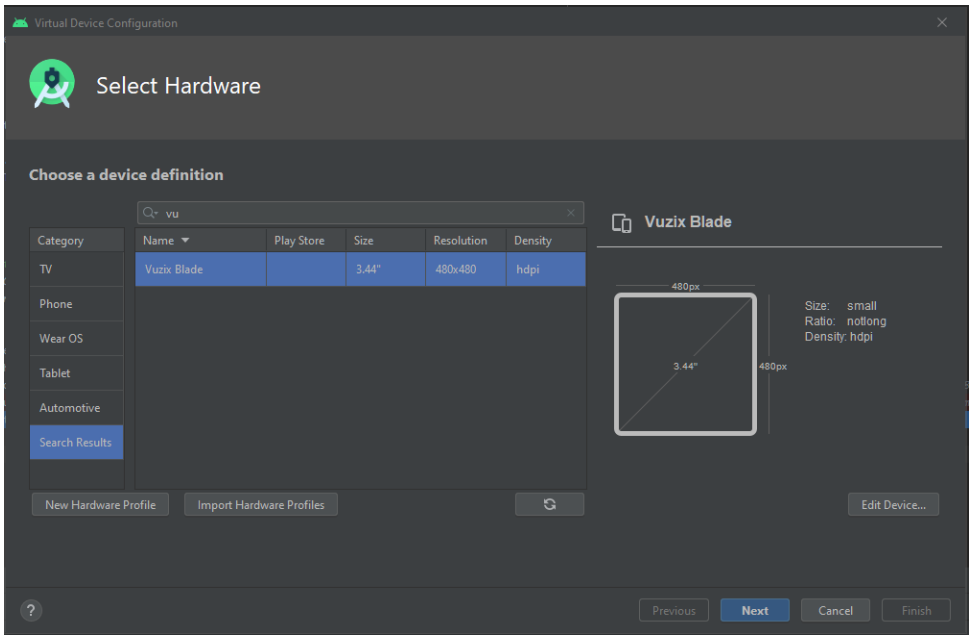
3. Select Import Hardware Profiles



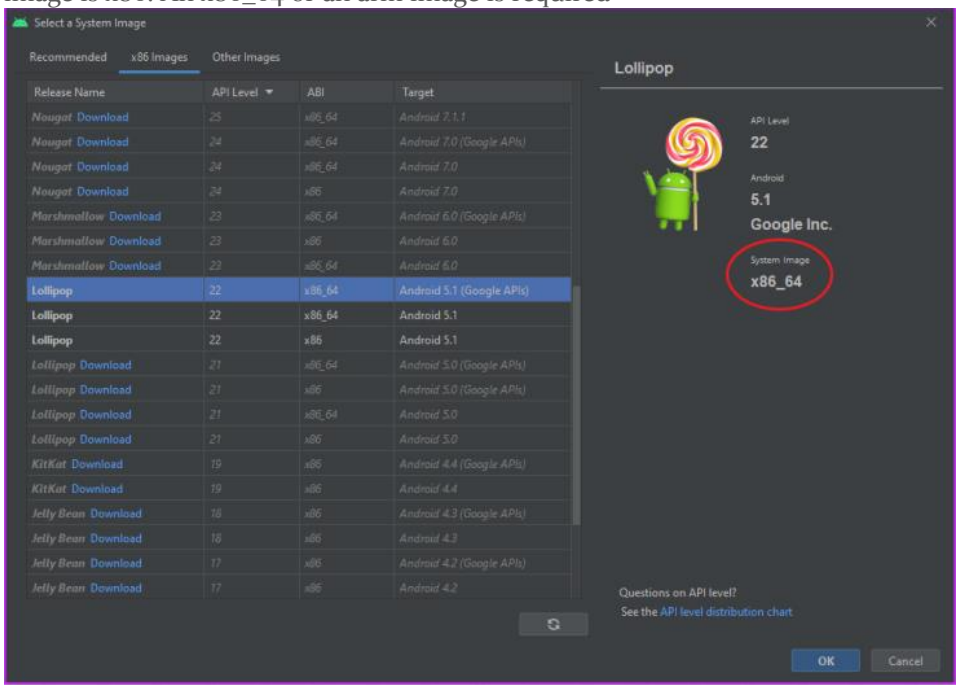
4. Select vuzix-blade_v3.xml from inside the project folder sdmay21-05



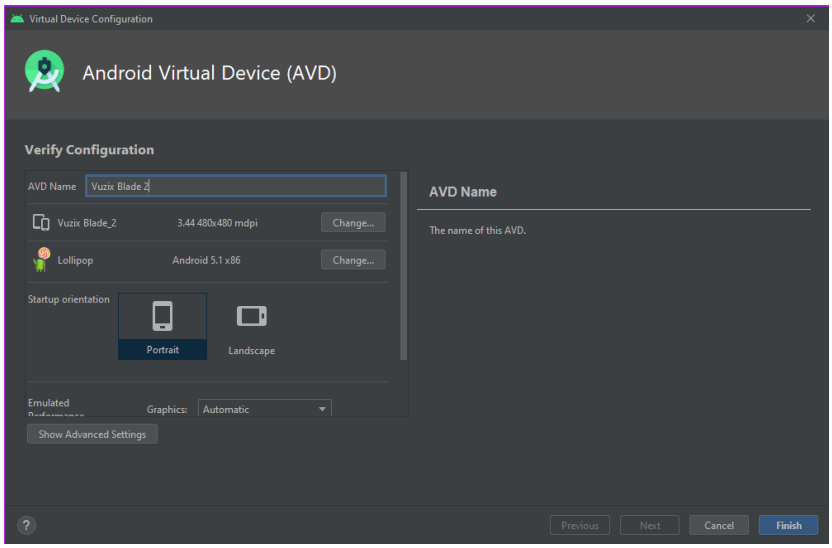
5. Vuzix Blade will now be an option for device definition. Select it and then press Next



6. Select system image: Lollipop/API 22. System image will not work with Stockfish if the image is x86. An x86_64 or an arm image is required



7. Check information is correct and select Finish

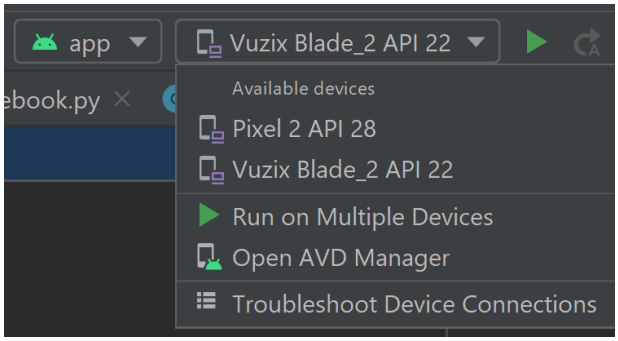


Building and Deploying to the Actual Device

1. Power on the Vuzix Blade
2. Navigate to Settings -> Developer Options.
3. Enable File Transfer and Developer Modes

a. You many also want to turn on Android Debug Bridge (ADB) however this is not required.

4. Connect the Vuzix Blade to your computer using a Micro-USB cable such as the one provided in the box
    a. Note: The cable in the box is a bit of a pain to work with, and we recommend users get a longer cable that they know works
5. The device should now be selectable from the list of available devices as shown below:



    a. Occasionally the device may not appear here. When this happens, we recommend users click “Troubleshoot Device Connections” in the above photo and follow the recommended steps.

6. Once you have selected the device, you can now press the Green Play Button next to the device name (see picture above) to deploy the application to the Vuzix Blade
    a. Alternatively, users may deploy the app using ADB Wireless Debugging. However, we did not do this and recommend users physically connect the Vuzix Blade to the computer and deploy using the Micro-USB. For those that do need or want to use ADB Wireless Debugging, please consult the following: https://developer.android.com/studio/command-line/adb

Android Code Documentation

We went through and commented the Android Code to include documentation information we believe would be useful for developers to know. However, at a high level, the project has the following classes:

- Camera Preview Class: Basic Camera Preview Class that gets utilized in the Gameplay class to display the camera preview to users.
- Difficulty Menu: Menu that allows a user to change the difficulty of the stock engine.
- Gameplay Activity: Responsible for the Core Gameplay loop of capturing and image, passing it to the CV module, and printing the recommended move
- Image Capture Activity: Was used to take a picture but has been replaced with the Gameplay Activity
- Settings: Activity that allows you the change settings in the app. Currently the only setting that we have are difficulty
- StockfishUtil: Class for communicating with the stockfish engine
- Welcome Activity: Initial Activity that the user sees when opening the app. This is the main menu and allows the user to get into core gameplay or settings.

Computer Vision Installation

To start development work on the computer vision pipeline, a more complex setup is required than for Android Development. Numerous libraries and programs need to be installed before this development can begin. The process for installing each of these libraries and programs is discussed in its own subsection below.

Python

The current version of Chaquopy uses Python V3.8.X and therefore, we recommend developers install this version of Python when developing the pipeline. This can be done by:

1. Vist https://www.python.org/downloads/ and download Python V3.8.X that corresponds to your system.
2. Follow the setup instructions.
3. You may need/want to add Python to your path as well as it greatly simplifies the commands for installing the other libraries
4. Dependencies can be installed by running the command “pip install –r requirements.txt” from the computer vision directory, or by the steps listed below.

NumPy

1. Open your command Prompt
2. Enter “pip install NumPy.” Note that you may need to use the full path to pip if you did not add Python to your path variable

OpenCV

1. Open your command Prompt
2. Enter “pip install OpenCV-python.” Note that you may need to use the full path to pip if you did not add Python to your path variable.

Stockfish

1. Open your command Prompt
2. Enter “pip install stockfish.” Note that you may need to use the full path to pip if you did not add Python to your path variable.

Jupyter Labs

1. Open your command Prompt
2. Enter “pip install jupyterlab.” Note that you may need to use the full path to pip if you did not add Python to your path variable.
3. Add the path to jupyter lab to your path variable
4. From the current command prompt window or any new command prompt you can now launch jupyter labs by entering “jupyter lab” to launch jupyter lab in your current browser window.
5. Navigate to the directory containing the computer vision files in the new window open in your browser.

Computer Vision Settings

The settings for the computer vision module are generally controlled by the configuration file. This file is the first place developers should turn when looking to make general operating changes to the pipeline. The file is well commented, and explanations should be given for what each variable controls and how changes affect the program. If the changes you seek are not possible by making changes to the configuration file, developers should then look into the CompleteNotebook.py file and see if there exists current functionality to get the desired settings and performance they seek. If not, custom changes can be added to both the configuration file and the CompleteNotebook.py file in order to achieve this performance.

Computer Vision Code Documentation

We went through and commented the CompleteNotebook.py to include documentation information we believe would be useful for developers to know. However, at a high level the project has the following functions:

- setup_scanning
    - Runs through some of the pipeline to determine what color pieces are in the front row of cells closest to the user.
    - The player's color is determined by which color has 5 or more pieces detected.
    - If neither color has 5 or more pieces, successful is returned as False.
- find_board_corners
    - Finds the corners of the board using green circles for corner designators.
    - Returns the corners arranged in clockwise order from the close left corner and an accuracy metric to determine if all 4 corners were found (returns corners, accuracy).
- shift_perspective
    - Shifts the board from a 3d perspective to a 2d top-down perspective in a square.
    - Sets this image to the Img_Provider image.
    - The goal of this is to make the rest of the computer vision logic more straightforward and easier since only the top-down view is worried about.
- get_cell_from_index
    - Crops the board image to the cell specified by the index coordinates.
    - Assumes the provided image is cropped to only contain the board, and that the top left cell is A8 from the perspective of the player.
    - Get a pair of (col, row) values used to select a cell
    - Returns an image cropped to the specific cell of the board
- check_cell
    - Given a cell image, like from get_cell_from_index, determines whether the cell is occupied by a piece.

    - If it is occupied, determine whether the piece is white or black.
    - Returns 0 if no occupied, -1 if dark piece, 1 if light piece
- get_board_representation
    - Gets the current occupancy status of the spots on a board. Spots will be represented by a 0 if not occupied, -1 if occupied dark, 1 if occupied light. The flow starts by trying to find the board corners. If four corners are found the board is perspective shifted so that it represented in 2d. Finally, using this perspective, the occupancy of the board is calculated.
    - Returns a 2-dimensional array accessed in the form [column][row] where each spot represents a square on the board.
- get_board_diffs
    - Calculates a board containing the difference between 2 board states. Spots are represented by 0 meaning no difference, -1 being a move from that square, and 1 being a move to that square.
    - Returns an 8 by 8 chessboard with individual cell changes along with the number of differences total (board, diffs).
- get_last_move
    - Finds the most recent move in UCI notation
        - Ex: from the starting position, moving the pawn at d2 to the empty cell d4 would be "d2d4"
    - Diffs: the differences in the board states; should only be two cells if we ignore edge cases
    - Returns a string representation of the most recent move in UCI notation
        - Ex: "e2e3", "b7b6", etc.
- blade_scan
    - Runs the CV pipeline on inputs from the Vuzix Blade.
    - Runs like a server where it constantly loops through the pipeline. The pipeline flows as follows:
        - Calibration loop
            - Load image
            - Calibrate corner color
            - Calibrate red/open squares
            - Calibrate piece colors
        - Setup board initial state
        - Main pipeline loop
            - Load image
            - Find corners
            - Warp board perspective
            - Check each cells occupancy (open, light, dark)
            - Compare current board to previous board to get move
            - Report the move to the FILE_WRITE_MOVE file

USING THE VUZIX BLADE

1. If you have just deployed the Game Advisor Application from Android Studio, the application should open automatically. However, for those that are not installing the app for the first time, you should scroll through the home page to find the Game Advisor Application and select it by tapping the touch pad on the right side of the glasses as if you were tapping your temple.

2. Once the application is launched, users should select “Start Game” to begin playing.
3. Once the user sees a camera preview, the glasses are ready to accept photos of the chess board. To capture photos, please scroll to “Capture Image” using the touchpad on the glasses and tap this touchpad to capture the select this button and capture the image.
4. The Game Advisor Application will now process the image and return a message to the user indicating the recommended move at which point the user should execute it. In some scenarios, the image captured may be poor, and therefore we cannot accurately determine the board state. In this case, the user will be prompted with an error message and should take another picture.
5. Steps 3 and 4 should be repeated until the game is completed.

Testing the Computer Vision Pipeline

1. Open jupyter labs by running “jupyter lab” from your terminal.
    a. If you have not isntalled jupyter lab please visit the Jupyter Lab Section in Appendix III for instructions on how to do so.
    b. Note you may need to use the full path if jupyter labs was not added to your path variable after it was installed.
2. Navigate to the Computer Vision Folder in the projects Git Repository.
3. Select the Notebook you wish to run. The testing notebook is used to test certain functionality while the Complete Loop Notebook has very detailed and comprehensive printouts that are great for debugging and testing the entire pipeline.
    a. If you wish to run the full loop notebook in debug/test mode, you may need to change the configuration file to allow printouts at different stages of the pipeline.
4. Verify the pipeline passes the tests by visual inspection or through reports where applicable.
5. Run test_CV.py in this Computer Vision Folder from command line to test the functions not revolving around images (getboarddiffs & getlastmove)

Next Steps

Accuracy Improvement and Reducing Assumptions

We were able to develop a computer visions pipeline that can semi accurately create a board state from images. In doing so, we had to make a variety of simplifications and assumptions, including a known initial state, adding red dots to each board square, adding green dots to the board corners, and requiring traditional board sets. These simplifications and assumptions made the project much easier for us to implement as a proof of concept; however, they do not allow for full practical usage. For the next senior design or independent study group that takes over the project, there is clearly work to be done to remove some of these assumptions and simplifications. The first two things the next group or individual should work to remove are the red and green dots. These were added to simplify detection capabilities by allowing us to determine space occupancy and the board's location based on the consistent location and sharp contrast of the dots in relation to the board. However, since actual chess boards do not have these dots, detection capability needs to be modified and improved to not rely on them. One potential solution is to look in the center of each cell for a color that matches the color of a standard black/white piece rather than the red color of the dot. The detection and framing of the board are likely to be a more complicated problem, and we recommend that users look into line detection as a potential way to solve this problem.

Once the dots are removed, ways to remove the remaining assumptions and simplifications may be explored. However, these assumptions will prove substantially more difficult. We restricted the gameplay of the application to start from the initial state so that we did not have to do piece identification, but if this assumption is removed, the pipeline will now need to not only detect if a space is occupied but by what piece. This will likely require training an ML model to recognize the pieces for each board cell that is occupied, at least on the first image captured. From there, the location of each piece would be known, and the current pipeline could be used to detect changes in the board state. Additionally, the computer vision could be expanded to include nontraditional chess sets such as Game of Thrones. However, doing so is likely overkill unless the product is trying to be released and sold publicly.

In addition to removing the assumptions and simplifications, there is additional work to be done to increase the accuracy of our board state determination. Currently, the detection only works in ideal scenarios with minimal shadows and glare. We started a calibration process to help combat some of these issues; however, this process is not perfected completely and could still be fine-tuned to help deal with these scenarios. Additionally, there are scenarios where a piece is hidden from view, such as a queen block a pawn, that need to be handled. Currently, we require the wearer of the glasses to take another picture in this scenario, but logic could be implemented for this scenario to recognize this hidden piece in our board state. We recommend that the next group start making improvements to the pipeline as it relates to accuracy and consistency before moving on to removing assumptions and simplifications.

Expansion to Other Games

In addition to making improvements to the chess advisor, the next group of developers could move to implement other tabletop games that follow a similar flow. These games could include basic games such as Tic-Tac-Toe to more complex games such as connect-four and checkers depending on the developer's interests and skill levels. When designing and developing the game advisor application, we were as modular as we could be, allowing the next developers to reuse many of the existing classes, functions, and activities in non-chess implementations.