AR Chess Advisor

DESIGN DOCUMENT

Team Number 05

Dr. Zambreno — Faculty Advisor
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

Revised: 11/15/2020 Version 3

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
    - Colors allow for usage by colorblind individuals
- Mobile/Android
    - Clear layout and easy navigation
    - Flexible components allowing for customized process
    - Responsive, adaptive, and iterative
- Human-Centered

Software Development Standards/Practices:

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

    - Modularity
    - Naming Conventions
    - Comments
- Agile Methodology Standards
- Lean Development – fail-fast

Engineering Standards:

- Reliability
- Scalability
- Performance

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 high 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:

- 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 185: Intro to Problem Solving I
- CPR E 186: Intro to Problem Solving II: Project
- CPR E 230: Cyber Security Fundamentals
- CPR E 308: Operating Systems, Principles, and Practice
- CPR E 388: Mobile Platforms
- CPR E 530: Network Protocols and Security
- 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

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
- Jupyter Notebooks for Iterative Development

Table of Contents

List of figures/tables/symbols/definitions

List of Figures

List of Tables

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 high contrast to allow for consistent board and state detection.

The corresponding backend running the game (chess) engine may 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 long 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

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 high 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

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 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 his 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.

Limitations:

The project should not exceed $1000. The budget is mainly to be used 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 processing. The game engine API allows us to set a maximum time to calculate a move. Still, additional time may be needed for computer vision processing and communication between the headset and mobile device (if necessary). Specific time limitations are to be determined.

1.7 Expected End Product and Deliverables

Our project will consist of four main pieces: An AR glasses front end, a computer vision back end system, a strategy board game engine back end system, and potentially an Android companion app. These four 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 is a multitude of options for AR glasses and game engines to choose from, and decisions need to be made in order 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 game pieces. 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 – mid-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 a full 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 will be smooth and clean; computer vision will be able to recognize game pieces and board clearly. 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 Task Decomposition

The project is broken down into research and design, and development:

Research and Design:

1. Generate Requirements
    1.1. Meet with Product Owner and Advisor Dr. Zambreno for project kickoff
    1.2. Research the problem and empathize with users
2. Market Research
    2.1. AR Glasses
        2.1.1. Depends on 1.2
    2.2. Computer Vision Software/Libraries
        2.2.1. Depends on 1.2
    2.3. Game Engines
        2.3.1. Depends on 1.2
    2.4. Network Communication Protocols
        2.4.1. Depends on 1.2
    2.5. Existing Mobile Back-ends
        2.5.1. Depends on 1.2
3. Define Solution
    3.1. Select AR Glass Device
        3.1.1. Depends on 2.1
    3.2. Select Computer Vision Software and Libraries
        3.2.1. Depends on 2.2
    3.3. Select Game Engine
        3.3.1. Depends on 2.3
    3.4. Select where the backend will live
        3.4.1. Make an Ideal Selection
            3.4.1.1. Depends on 2.3, 3.1
        3.4.2. Determine backup selection
            3.4.2.1. Depends on 2.3, 3.1

Development:

1. AR Glass Application
    1.1. Project Creation
        1.1.1. Create the project within Android Studio and select Design Layout and Navigation Scheme
            1.1.1.1. Depends on: N/A
        1.1.2. Import the Vuzix Blade Hardware Profile
            1.1.2.1. Depends on 1.1.1
    1.2. Welcome Activity
        1.2.1. Create Welcome Activity to be seen upon Glass Power On

1.2.1.1. Depends on 1.1
1.2.2. Connect to saved paired devices. This may not be necessary if the entire app can run on the Vuzix Blade.
1.2.2.1. Depends on 1.1, 1.2.1, 1.3.2
1.3. Initial Setup
1.3.1. Develop Activities for device setup, including potentially pairing with a companion device if necessary
1.3.1.1. Depends on 1.1
1.3.2. Create a way to save paired devices so that once initially paired, devices are auto paired upon following boot ups. This may not be necessary if the entire app can run on the Vuzix Blade.
1.3.2.1. Depends on 1.1, 1.3.1
1.4. Calibration
1.4.1. Develop Activities needed for camera and computer vision algorithm calibration
1.4.1.1. Depends on 1.1.2, 1.5.1, 1.3.1
1.5. Image Capture
1.5.1. Develop Activities needed for capturing an image
1.5.1.1. Depends on 1.1.2, 1.3.1
1.5.2. Setup Voice Commands to take a picture on user command
1.5.2.1. Depends on 1.1.2, 1.3.1
1.5.3. Setup Touch sensors to take a picture on a user gesture
1.5.3.1. Depends on 1.1.2, 1.3.1
1.6. Pre-processing
1.6.1. Convert color images to grayscale to minimize computational complexity and data size
1.7. Display Move
1.7.1. Develop Activity for Displaying the engine's recommended move to the user
1.7.1.1. Ideally, through an AR overlay, but text is sufficient
1.7.1.2. Depends on 1.1.2
1.8. Communication
1.8.1. Develop, import, or customize the library for connecting to WIFI (or any other network medium we will need to connect to the backend). This may not be necessary if we run everything on the glasses.
1.8.1.1. Depends on 1.1.1
1.8.2. Develop, import, or customize the library for encoding images to be passed to the backend
1.8.2.1. Depends on 1.1.1
1.8.3. Develop, import, or customize library for encoding and decoding text passed to or from the backend
1.8.3.1. Depends on 1.1.1
2. Backend
2.1. Project Creation
2.1.1. Create the project within Android Studio and select Design Layout and Navigation Scheme
2.1.1.1. Depends on: N/A

2.2. Communication
- 2.2.1. Develop, import, or customize library for connecting to WIFI (or any other network medium we will need to connect to the front-end). This may not be necessary if we run everything on the glasses.
  - 2.2.1.1. Depends on 2.1.1
- 2.2.2. Develop, import, or customize the library for decoding images passed from the front end
  - 2.2.2.1. Depends on 2.1.1
- 2.2.3. Develop, import, or customize library for encoding and decoding text passed to or from the front-end
  - 2.2.3.1. Depends on 2.1.1
- 2.2.4. Develop, import, or customize the library for communicating with the Game Engine
  - 2.2.4.1. Depends on 2.1.1

2.3. Computer Vision
- 2.3.1. Initial Setup
  - 2.3.1.1. Ensure we can receive images transmitted by the front-end
    - 2.3.1.1.1. Depends on 1.8.1, 2.1, 2.2.1
  - 2.3.1.2. Create dataset for use in Board State Determination (May not be explicitly required if we do not use nn/ml)
  - 2.3.1.3. Setup codebase location in GitLab
  - 2.3.1.4. Set up OpenCV project-specific .gitignore
    - 2.3.1.4.1. Depends on 2.3.1.3
- 2.3.2. Calibration
  - 2.3.2.1. Ensure the images we receive are in a standardized format (file type, size, orientation, etc.)
    - 2.3.2.1.1. Depends on 2.3.1.1
- 2.3.3. Image Processing
  - 2.3.3.1. Setup python notebook and Generic OpenCV IO
    - 2.3.3.1.1. Depends on 2.3.1.3
  - 2.3.3.2. Ensure we can do edge detection of an image
    - 2.3.3.2.1. Depends on 2.3.3.1
- 2.3.4. Board State Determination
  - 2.3.4.1. Do line detection
    - 2.3.4.1.1. Depends on 2.3.3.1
  - 2.3.4.2. Check for chessboard layout from detected lines
    - 2.3.4.2.1. Depends on 2.3.4.1
  - 2.3.4.3. Only highlight chessboard after detecting layout
    - 2.3.4.3.1. Depends on 2.3.4.2
  - 2.3.4.4. Ensure we can find a single chess piece and place a bounding box around it
    - 2.3.4.4.1. Depends on 2.3.2.1
  - 2.3.4.5. Extend to find multiple chess pieces and place a bounding box around them
    - 2.3.4.5.1. Depends on 2.3.4.4
  - 2.3.4.6. Detect the type of each chess piece found
    - 2.3.4.6.1. Depends on 2.3.4.4
  - 2.3.4.7. Detect the position of each chess piece on the board.

2.3.4.7.1. Depends on 2.3.4.4, 2.3.4.5

2.3.4.8. Ensure we have and use a set format for the board state when communicating with the game engine

2.4. Game Engine

2.4.1. Game Engine Setting Customization

2.4.1.1. Develop Activities to change the game engine settings for items such as difficulty and response time (depth in chess engines)

2.4.1.1.1. Depends on 2.1.1

2.4.2. Import Game Engine

2.4.2.1. Download Game Engine

2.4.2.1.1. Depends on: N/A

2.4.2.2. Build and Compile the Game Engine Binary

2.4.2.2.1. Depends on 2.4.2.1

2.4.2.3. Import the Binary into the Back-end. Do any necessary environment configuration as needed.

2.4.2.3.1. Depends on 2.4.2.2, 2.1.1

2.4.3. Result Decoding

2.4.3.1. Develop, import, or customize the library for decoding the recommended move into terminology the average user can understand.

2.4.3.1.1. Depends on 2.1.1, 2.4.2.2

2.2 Risks And Risk Management/Mitigation

The possible risks, their probability, and the mitigation strategies for each are listed below:

- AR Glass Device is not capable of running the front-end and backend modules together – 40%
    - To minimize this risk, we will develop the backend end functionality modularly with as little cohesion to the front end as possible so that if necessary, we can easily port the code to a separate application. We will also begin developed early in Cpre 491 so that we have time to pivot if necessary.
- AR Glass Device does not have an emulator – 99%
    - To minimize this risk, we will select an AR Glass Device that runs Android so that all members can do development using the android emulators or an android mobile device.
- Vuzix Developer Account Requires a Fee – 20%
    - To minimize this risk, we created a developer account before purchasing the glasses and spent a week exploring and downloading different resources while looking for a paywall.
- Integrity: Attackers may be able to send fake requests to the backend or front-end components with false images or recommended moves – 40% (Odds AR Glass Device cannot run front-end and backend together)
    - This risk will be solved in implementation through encryption and hashing so that 1) The hacker cannot send false images or moves, and 2) The hacker cannot send invalid images or messages. This is only a risk if we must separate the front end and back end modules and use some form of wireless communication.

- We are unable to find viable chessboard and chess piece datasets for classifier/model training – 80%
    - We can create our own classifiers and/or models for recognizing the boards and pieces. Because this issue is so highly probable, we will implement the rest of our program with modularity in mind, allowing us to quickly switch to our own implementations if required.
- Python implementation of OpenCV runs too slow to be viable – 40%
    - We can reimplement the OpenCV processing in C/C++, which will run faster and use fewer system resources.
- Chess piece detection is too low to be viable for the final product – 70%
    - We can pivot to a game with less complex pieces/setup, such as checkers or connect four. This will allow us to keep the spirit of the product while being able to achieve a complete working product.
    - We may also need to implement a new game engine, depending on the complexity of the replacement game chosen.
    - This risk is minimized based on our assumptions that people will use traditional chess sets and the environmental requirement that the device is played in a well-lit environment.
- Inability to detect a state other than base starting state – 60%
    - This corresponds to the chess piece detection being too low for a viable product. If we cannot detect and determine the location of each piece, we have no way to determine an arbitrary initial state such that the users could put the glasses on mid-game. To limit this risk, we will assume that the glasses are not put on midgame, and therefore the initial board state is known.
- Visual Overlay is difficult to understand, and the feedback pipeline is difficult to use and unresponsive – 60%
    - Our product is designed to be extremely user friendly, including being usable for even the most basic and unfamiliar game players as well as those that are unfamiliar with technology. Therefore, our user interface and interaction need to be very friendly. To make the product as user friendly as possible, we will implement two different ways for capturing images (voice and touch). Additionally, we will use basic chess terminology for our prototype with the goal of a full AR overlay such that the user does not need any chess knowledge to understand the moves reported by the engine.

2.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

Key milestones for the project are based on our features, deliverables, and use cases. Each milestone is completed by the use case verification specified in section 5.

- Completed Market Research and get the AR Glasses Ordered no later than 9/18/20
    - This requires the Market Research and task 3.1 Selected AR Glass Device to be completed.
- Begin Proof of Concept Development no later than October 1, 2020
    - This requires all of section 3: Define Solution to be completed.

- Computer Vision Algorithms can detect chessboard and recognize a few pieces no later than the Prototype Demo at the end of the fall semester.
    - This requires that all Computer Vision subtasks up to and including Section 2.3.4.4 to have been completed by the end of the fall semester.
- Proof of Concept Development Complete by the end of the 2020 fall semester.
    - This requires that all Computer Vision subtasks up to and including Section 2.3.4.4 to have been completed by the end of the fall semester as well as AR Glass tasks (and their corresponding subtasks) 1.1 Project Creation, 1.2 Welcome Activity, 1.5 Image Capture, 1.6 Preprocessing, 1.7 Display Move, and 1.8 Communication.
- Minimum Viable Product Complete by mid-March 2021
    - This requires all enumerated tasks below to be complete and passed the initial round of interface and acceptance testing.
- Finalized Design Complete by the end of Spring 2021
    - This requires multiple iterations of development and user acceptance testing after completing the minimum viable product and should include nice to haves.

When developing metrics, is it important for our team to note that we will be using an Agile Development approach. Therefore, the metrics for each task will focus on completion status, accuracy, timing, and tests passed as these types of metrics are well suited for an iterative style of development.

Given the detail and the sheer number of tasks we broke the project down into, the table format used in this section as well as section 2.6 is not ideal. Our major project components are divided into tasks, each with many subtasks, and these subtasks also have subtasks when you get to the backend components. We did our best to represent the hierarchy using task numbers and a combination of fill colors and indentation. The hierarchy we are attempting to illustrate in the table for the top-level AR Glasses component is shown below in Figure 1.

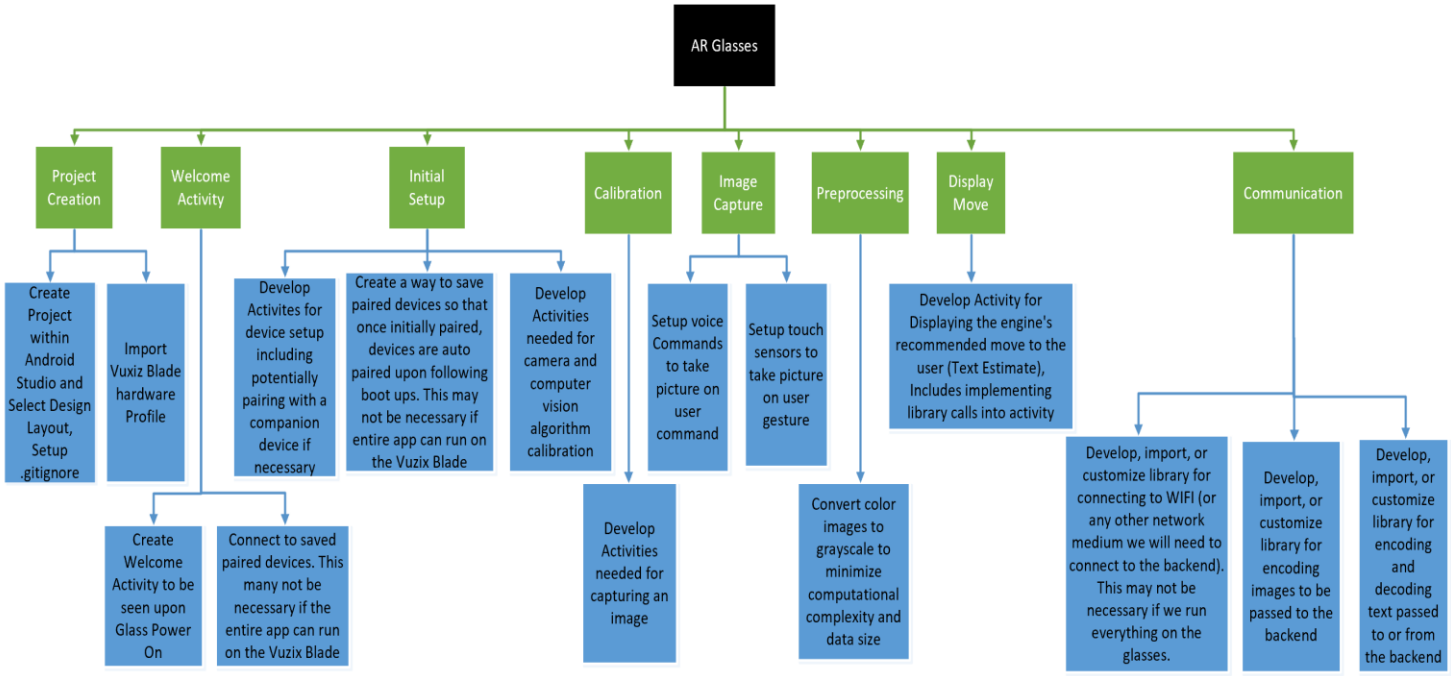

Figure 1: Example Table Hierarchy for the AR Glass Component

Table 1 below matches each task up to its respective metrics, using the task number in the far-left column as well as fill colors to try and illustrate the hierarchy of the tasks in a table format. For a much clearer representation of this table and task hierarchy, please consult the extremely detailed and extensive breakdown in the excel spreadsheet posted on our website.

Task Number	Top Level Module	Task Name	Subtask Description (If Needed)	Further Subtask Description (If Needed)	Task Metrics
1	Generate Requirements				
1.1		Meet with Product Owner and Advisor Dr. Zambreno for project kickoff			N/A
1.2		Research the problem and empathize with users			Number of areas researched, types of users explored, use cases determined
2	Market Research				Number of 2.X Subtasks Completed Out of 5

2.1		AR Glasses			Number of viable options found and explored
2.2		Computer Vision			Number of viable options found and explored
2.3		Chess Engine			Number of viable options found and explored
2.4		Network Communication and Protocols			Number of viable options found and explored
2.5		Existing Mobile Backends			Number of viable options found and explored
3	Define Solution				Number of 3.X Subtasks Completed Out of 4
3.1		Select AR Glass Device			Selected? (Y/n)
3.2		Select Computer Vision Software and Libs			Selected? (Y/n)
3.3		Select Chess Engine			Selected? (Y/n)
3.4		Select Where Backend Will Live			Number of 3.4.X Subtasks Completed out of 2
3.4.1			Make and Ideal Backend Selection		Selected? (Y/n)
3.4.2			Determine backup backend selection		Selected? (Y/n)
1	AR Glass Application				Number of 1.X Subtasks Completed Out of 8
1.1		Project Creation			Number of 1.1 Subtasks Completed out of 2
1.1.1			Create Project within Android Studio and Select Design Layout, Setup .gitignore		Completed? (Y/n)
1.1.2			Import Vuxiz Blade hardware Profile		Completed? (Y/n)

1.2		Welcome Activity			Number of 1.2 Subtasks Completed Out of 2
1.2.1			Create Welcome Activity to be seen upon Glass Power On		% Complete?# Screens Completed? Communication Calls hooked up?
1.2.2			Connect to saved paired devices. This many not be necessary if the entire app can run on the Vuzix Blade		Can we connect to one device? How many devices are saved that we try to connect to?
1.3		Initial Setup			Number of 1.3 Subtasks Completed Out of 2
1.3.1			Develop Activites for device setup including potentially pairing with a companion device if necessary		% Complete? # of Screens completed? Interfaced with communication library?
1.3.2			Create a way to save paired devices so that once initially paired, devices are auto paired upon following boot ups. This may not be necessary if entire app can run on the Vuzix Blade		Can we save a device? # of devices we save?
1.4		Calibration			Number of 1.4.X Subtasks completed out of 1
1.4.1			Develop Activities needed for camera and computer vision algorithm calibration		% Complete?# Screens Completed? Communication Calls hooked up? Able to send and recieve things with the computer vision module? (Y/n) What are we able to send and recieve currently? (Text, Images)

1.5		Image Capture			Number of 1.5.X Subtasks completed out of 3
1.5.1			Develop Activities needed for capturing an image		% Complete? # Screens completed? Communication Calls Hooked Up?
1.5.2			Setup voice Commands to take picture on user command		Able to recognize basic voice commands? (Y/n) How many custom commands does it respond to to take the picture? % complete?
1.5.3			Setup touch sensors to take picture on user gesture		Able to recognize basic gestures? (Y/n) % complete?
1.6		Pre-Processing			Number of 1.6.X Subtasks Completed out of 1
1.6.1			Convert color images to grayscale to minimize computational complexity and data size		Able to translate/recolor captured images to grayscale (Y/n)?
1.7		Display Move			Number of 1.7.X Subtasks Completed out of 1
1.7.1			Develop Activity for Displaying the engine's recommended move to the user (Text Estimate), Includes implementing library calls into activity		% Complete? # Screens completed? Communication calls hooked up? (Y/n)
1.8		Communication			Number of 1.8.X Subtasks Completed Out of 3
1.8.1			Develop, import, or customize library for connecting to WIFI (or any other network medium we		% complete? What stage are you in? (research, import, customize, or develop) Portability? Easy for

			will need to connect to the backend). This may not be necessary if we run everything on the glasses.		other activities to call methods in library?
1.8.2			Develop, import, or customize library for encoding images to be passed to the backend		% complete? What stage are you in? (research, import, customize, or develop) Portability? Easy for other activities to call methods in library?
1.8.3			Develop, import, or customize library for encoding and decoding text passed to or from the backend		% complete? What stage are you in? (research, import, customize, or develop) Portability? Easy for other activities to call methods in library?
2	Backend				Number of 2.X Subtasks Completed out of 4
2.1		Project Creation			Number of 2.1.X Subtasks Completed out of 1
2.1.1			Create the project within Android Studio and select the Design Layout and .gitignore		Completed? (Y/n)
2.2		Communication			Number of 2.2.X Subtasks Completed out of 4
2.2.1			Develop, import, or customize library for connecting to WIFI (or any other network medium we will need to connect to the front-end). This may not be necessary if we run everything on the glasses.		% complete? What stage are you in? (research, import, customize, or develop) Portability? Easy for other activities to call methods in library?

2.2.2			Develop, import, or customize library for decoding images passed from the front end		% complete? What stage are you in? (research, import, customize, or develop) Portability? Easy for other activities to call methods in library?
2.2.3			Develop, import, or customize library for encoding and decoding text passed to or from the front-end		% complete? What stage are you in? (research, import, customize, or develop) Portability? Easy for other activities to call methods in library?
2.2.4			Develop, import, or customize library for communicating with the game engine		% complete? What stage are you in? (research, import, customize, or develop) Portability? Easy for other activities to call methods in library?
2.3		Computer Vision			Number of 2.3.X Subtasks Completed out of 4
2.3.1			Initial Setup		Number of 2.3.1.X Subtasks Completed out of 4
2.3.1.1				Ensure we can receive images transmitted by the front-end	% complete? What stage are you in? (research, import, customize, or develop) Portability? Easy for other activities to call methods in a library?
2.3.1.2				Create a dataset for use in board state determination (May not be explicitly required if we do not use nn/ml)	% complete? What stage are you in? (research, import, customize, or develop) Portability? Easy for other activities to call methods in a library?
2.3.1.3				Set up codebase location in team Gitlab repository	Completed? (Y/n)
2.3.1.4				Set up OpenCV .gitignore	Completed? (Y/n)

2.3.2			Calibration		Number of 2.3.2.X Subtasks Completed out of 2
2.3.2.1				Ensure the images we receive are in a standardized format (file type, size, orientation, etc.)	Completed? (Y/n)
2.3.3			Image Processing		Number of 2.3.3.X Subtasks completed out of 3
2.3.3.1				Setup python notebook and Generic OpenCV IO	Completed? (Y/n)
2.3.3.2				Implement generic edge/contour detection of an image	% complete? What stage are you in? (research, import, customize, or develop) Portability? Easy for other activities to call methods in a library?
2.3.4			Board State Determination		Number of 2.3.4.X Subtasks Completed out of 8
2.3.4.1				Do line detection	% complete? What stage are you in? (research, import, customize, or develop) Portability? Easy for other activities to call methods in a library?
2.3.4.2				Check for chessboard layout from the detected lines	% complete? What stage are you in? (research, import, customize, or develop) Portability? Easy for other activities to call methods in a library?
2.3.4.3				Implement highlighting only the chessboard for testing and visualization purposes	% complete? What stage are you in? (research, import, customize, or develop) Portability? Easy for other activities to call methods in a library?
2.3.4.4				Ensure we can find a single chess piece and place a bounding box around it	% complete? What stage are you in? (research, import, customize, or develop) Portability? Easy for

					other activities to call methods in a library?
2.3.4.5				Ensure we can find multiple chess pieces and place accurate bounding boxes around them	% complete? What stage are you in? (research, import, customize, or develop) Portability? Easy for other activities to call methods in a library?
2.3.4.6				Detect the type of each chess piece found	% complete? What stage are you in? (research, import, customize, or develop) Portability? Easy for other activities to call methods in a library?
2.3.4.7				Detect the position of each chess piece on the board	% complete? What stage are you in? (research, import, customize, or develop) Portability? Easy for other activities to call methods in a library?
2.3.4.8				Ensure we have and use a set format for the board state when communicating with the game engine	Completed? (Y/n)
2.4		Game Engine			Number of 2.4.X Subtasks completed out of 3
2.4.1			Game Engine Setting Customization		Number of 2.4.1.X Subtasks completed out of 1
2.4.1.1				Develop activities to change the game engine settings for items such as difficulty and response time (depth in chess engines)	Settings understood? (Y/n) How many settings are we going to customize? % Complete? # of Screens Developed? # of customized setting completed?
2.4.2			Import Game Engine		Number of 2.4.2.X Subtasks Completed out of 3
2.4.2.1				Download Game Engine	Completed? (Y/n)

2.4.2.2				Build and Compile the Game Engine Binary	Completed? (Y/n)
2.4.2.3				Import the binary into the backend. Do any necessary environment configuration as needed	Imported? (Y/n) % Configured?
2.4.3			Result Decoding		Number of 2.4.3.X Subtasks complete out of 1
2.4.3.1				Develop, import, or customize the library for decoding the recommended move into terminology the average user can understand	% complete, Number of pieces translated? Number of locations translated?

Table 1: Individual Task Metrics

2.4 Project Timeline/Schedule

Our Gantt chart(s) are extremely detailed and contain a breakdown for each task. However, do the size of said chart; it is unable to fit within this document and is attached as a separate excel spreadsheet document. Therefore, we have compressed subtasks to fit under major tasks and broken months down into 4 segments rather than by individual days to generate compressed figures that can be used for this report. Even these charts are large, and we will have separate figures for milestones, project planning, and development below. Figure 2, below, contains the milestones discussed in section 2.3.

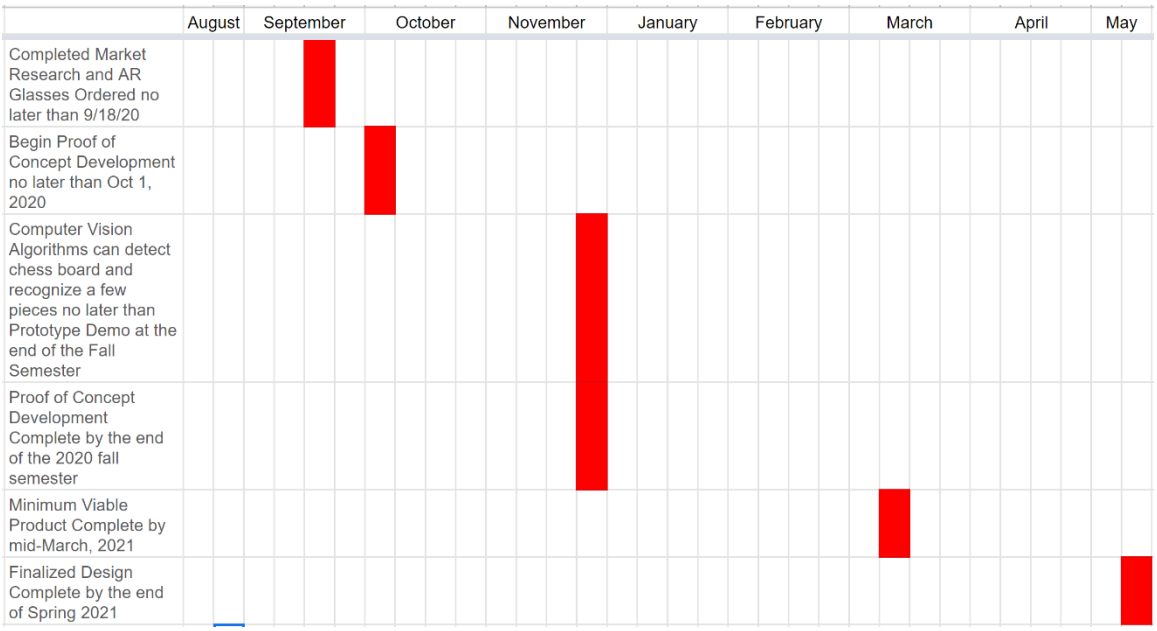

Figure 2: Gantt Chart – Milestones

Figure 3, below, contains the compressed Gantt Chart for Project Planning and Design. Note that as we encounter issues, the design may need to be revisited, and therefore these phases may be repeated throughout the semester.

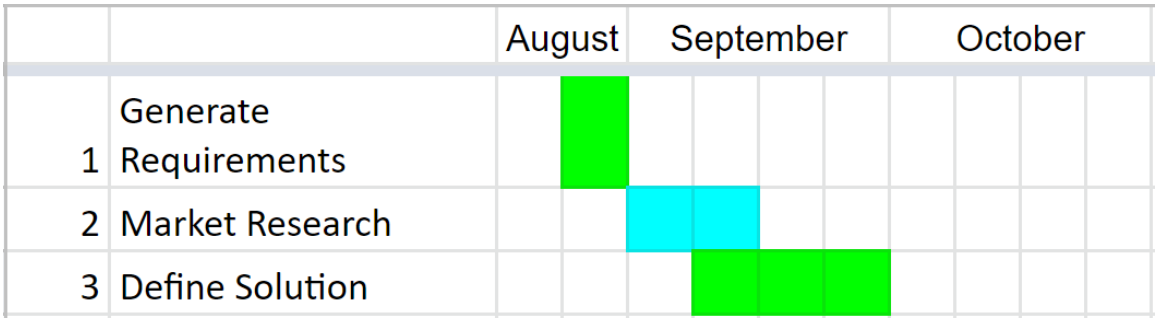

Figure 3: Gantt Chart - Project Planning and Design

Figure 4, below, is the compressed Gantt Chart for development. This chart initially appears to be more Waterfall than Agile due to the initial dependencies required. However, as we progress in the project, and especially during 492, we will be using more Agile principles and practices. At this point, we do not know what features we will add after initial development ends in February, so we simply added a section for adding features and testing.

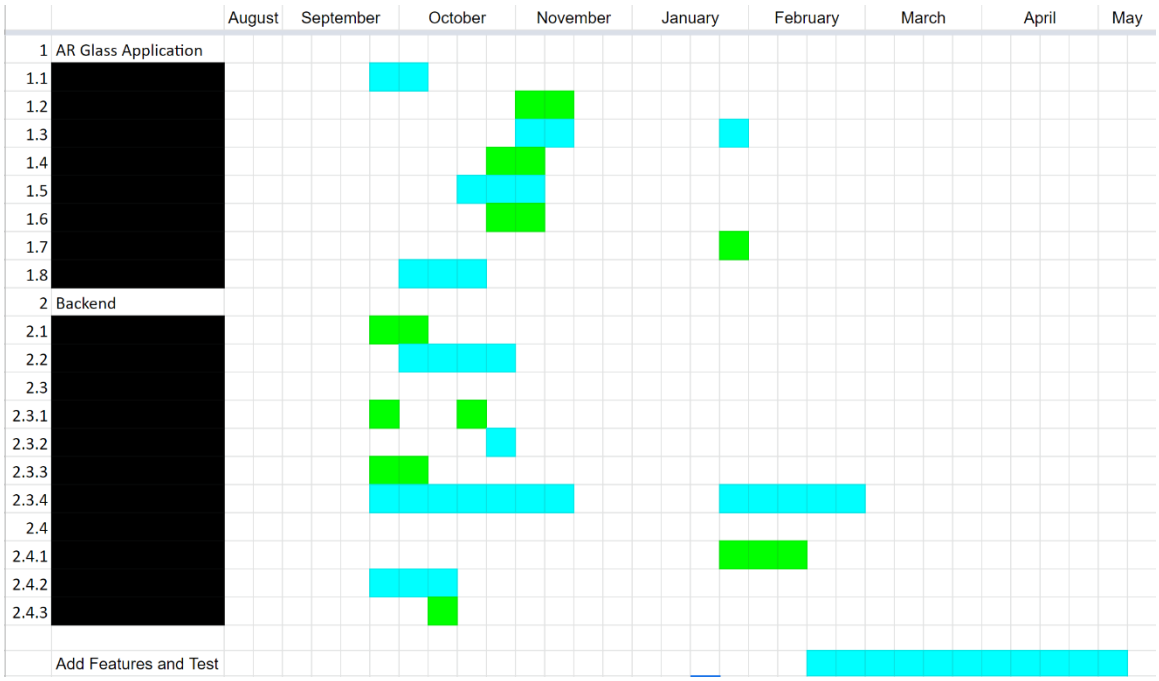

Figure 4: Gantt Chart - Development

2.5 Project Tracking Procedures

Our group is utilizing several different tools to track progress and ensure good communication. Our primary method of keeping in touch 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 will use GitLab to store and track our progress on development work overtime. Lastly, our group will use Trello boards for monitoring tasks during our sprints. While our initial development is more waterfall, this is necessary to get the project off the group, and we will transition entirely to the agile methodology after the initial baseline has been created. Regardless of what methodology (or mix) we are currently using, the Trello board will be significantly easier to track tasks than google drive.

2.6 Personnel Effort Requirements

As mentioned in section 2.3, given the detail and the sheer number of tasks we broke the project down into, the table format used in this section as well as section 2.3 is not ideal. Our major project components are divided into tasks, each with many subtasks, and these subtasks also have subtasks when you get to the backend components. We did our best to represent the hierarchy using task numbers and a combination of fill colors and indentation.

Table 2, below, shows the detailed personnel effort requirements for each task to complete our initial development. After these tasks are completed in Mid-February 2021, we will begin iterating and adding nice to have features. The total development hour estimate for the initial tasks is 409 hours. For a much clearer representation of this table and task hierarchy, please consult the extremely detailed and extensive breakdown in the excel spreadsheet posted on our website

Task Number	Top Level Module	Task Name	Subtask Description (If Needed)	Further Subtask Description (If Needed)	Hour Effort Estimate
1	Generate Requirements				
1.1		Meet With Product Owner and Advisor Dr. Zambreno for project kickoff			7
1.2		Research the problem and empatize with users			13
2	Market Research				
2.1		AR Glasses			12
2.2		Computer Vision			12
2.3		Chess Engine			12
2.4		Network Communication and Protocols			12
2.5		Existing Mobile Back-ends			12
3	Define Solution				
3.1		Select AR Glass Device			7
3.2		Select Computer Vision Software and Libs			30
3.3		Select Chess Engine			4
3.4		Select Where Backend Will Live			
3.4.1			Make and Ideal Backend Selection		7

3.4.2			Determine backup backend selection		7
1	AR Glass Application				
1.1		Project Creation			
1.1.1			Create Project within Android Studio and Select Design Layout, Setup .gitignore		3
1.1.2			Import Vuxiz Blade hardware Profile		1
1.2		Welcome Activity			
1.2.1			Create Welcome Activity to be seen upon Glass Power On		2
1.2.2			Connect to saved paired devices. This many not be necessary if the entire app can run on the Vuzix Blade		5
1.3		Initial Setup			
1.3.1			Develop Activities for device setup including potentially pairing with a companion device if necessary		7
1.3.2			Create a way to save paired devices so that once initially paired, devices are auto paired upon following boot ups. This may not be necessary if entire app can run on the Vuzix Blade		3
1.4		Calibration			
1.4.1			Develop Activities needed for camera and computer vision algorithm calibration		10
1.5		Image Capture			

1.5.1			Develop Activities needed for capturing an image		4
1.5.2			Setup voice Commands to take picture on user command		10
1.5.3			Setup touch sensors to take picture on user gesture		2
1.6		Pre Processing			
1.6.1			Convert color images to grayscale to minimize computational complexity and data size		5
1.7		Display Move			
1.7.1			Develop Activity for Displaying the engine's recommended move to the user (Text Estimate), Includes implementing library calls into activity		3
1.8		Communication			
1.8.1			Develop, import, or customize library for connecting to WIFI (or any other network medium we will need to connect to the backend). This may not be necessary if we run everything on the glasses.		7
1.8.2			Develop, import, or customize library for encoding images to be passed to the backend		3
1.8.3			Develop, import, or customize library for encoding and decoding text passed to or from the backend		2
2	Backend				
2.1		Project Creation			
2.1.1			Create the project within Android Studio and select		3

			the Design Layout and .gitignore		
2.2		Communication			
2.2.1			Develop, import, or customize library for connecting to WIFI (or any other network medium we will need to connect to the front-end). This may not be necessary if we run everything on the glasses.		7
2.2.2			Develop, import, or customize library for decoding images passed from the front end		3
2.2.3			Develop, import, or customize library for encoding and decoding text passed to or from the front-end		2
2.2.4			Develop, import, or customize library for communicating with the game engine		7
2.3		Computer Vision			
2.3.1			Initial Setup		
2.3.1.1				Ensure we can receive images transmitted by the front-end	4
2.3.1.2				Create a dataset for use in board state determination (May not be explicitly required if we do not use nn/ml)	4
2.3.1.3				Set up codebase location in team Gitlab repository	1
2.3.1.4				Set up OpenCV .gitignore	1
2.3.2			Calibration		

2.3.2.1				Ensure the images we receive are in a standardized format (file type, size, orientation, etc.)	2
2.3.3			Image Processing		
2.3.3.1				Setup python notebook and Generic OpenCV IO	3
2.3.3.2				Implement generic edge/contour detection of an image	3
2.3.4			Board State Determination		
2.3.4.1				Do line detection	2
2.3.4.2				Check for chessboard layout from the detected lines	5
2.3.4.3				Implement highlighting only the chessboard for testing and visualization purposes	2
2.3.4.4				Ensure we can find a single chess piece and place a bounding box around it	2
2.3.4.5				Ensure we can find multiple chess pieces and place accurate bounding boxs around them	5
2.3.4.6				Detect the type of each chess piece found	20
2.3.4.7				Detect the position of each chess piece on the board	20
2.3.4.8				Ensure we have and use a set format for the board state when communicating with the game engine	4

2.4		Game Engine			
2.4.1			Game Engine Setting Customization		
2.4.1.1				Develop activities to change the game engine settings for items such as difficulty and response time (depth in chess engines)	15
2.4.2			Import Game Engine		
2.4.2.1				Download Game Engine	1
2.4.2.2				Build and Compile the Game Engine Binary	3
2.4.2.3				Import the binary into the backend. Do any necessary environment configuration as needed	5
2.4.3			Result Decoding		
2.4.3.1				Develop, import, or customize library for decoding the recommended move into terminology the average user can understand	5

Table 2: Personnel Effort Requirements

2.7 Other Resource Requirements

The physical resources needed to complete this project would be

- A chessboard (gameboard) and chess (game) pieces: We will purchase 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 progresses, they will need to be more interaction between members of the team as components begin to integrate, and when testing is needed. This physical space could include one of our apartments but is likely to be a meeting space such as the TLA.

- Android Development Environment: The team will need an environment to develop the AR and Back-end Applications. The emulators in Android Studio should suffice for this.

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

2.8 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.

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 will use 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 should we pick. In this stage, we explore 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 approximately 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 Proposed 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 engines 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 camera calibration.
- We will likely use trained image classifier to locate and identify chess pieces on the chessboard.
- 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" application.

- 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 application, however, the backend may have to be hosted on a separate device due to processing limitations. 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 Environmental Requirements described in section 1.4, however, a simple information sheet included with the product explaining the recommended usage and storage environments could easily be included with the final product.

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 very 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 sperate out 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. So far, we have no reason to indicate that this will be necessary and can cut these modules if they are not needed.

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 was 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.

So far, we have had little information or observations to modify or iterate on our design. We did one iteration at the beginning of the semester, and the result of that iteration is described above. As we continue to progress throughout the project, we plan to use the observations through demos amongst the team and with Dr. Zambreno to generate ideas to modify or iterate over the design. Currently, the only iteration we have planned for CPRE 492 is to expand the move display to be an

AR overlay rather than just text. However, through demos over the next 6 months, we will have significantly more to iterate on once our prototype is completed.

3.6 Development Process

Our team has decided to use an Agile development approach for our AR Chess Advisor project. By following this approach, we will be able to manage our time best and ensure we are working efficiently towards our client's goals. Initially, our development will 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 will be positioned to transition to a modified Scrum.

We will have two scheduled meetings each week, where we give status updates and plan what work needs to get done for the coming "sprint." One meeting each week will include our client, allowing us to update them on our progress and receive regular feedback. This model will allow us to continually modify our tasks and pivot our focus areas as the project progresses in order to meet the client's needs best.

We will continue to use the Agile approach throughout the duration of the project, though the weekly meetings will likely go 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 plan to host the front end and back end components entirely on the AR Glasses as of now. This greatly simplifies our design's architecture as the interaction among modules is all contained within the same device. However, the proposed design does include a safety net of communication libraries and modules should the backend be hosted on a different device.

This section will showcase 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 5.

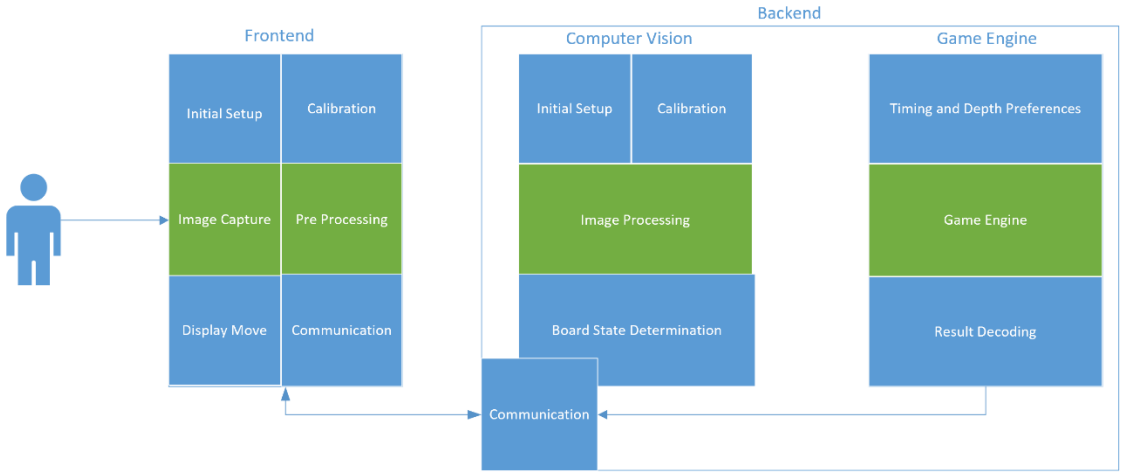

Figure 5: Modular Design Diagram

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 very 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 very 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 does not appear like it will be a problem, but we will track it as we go.

In addition to the system design sketch, Figure 6, 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.

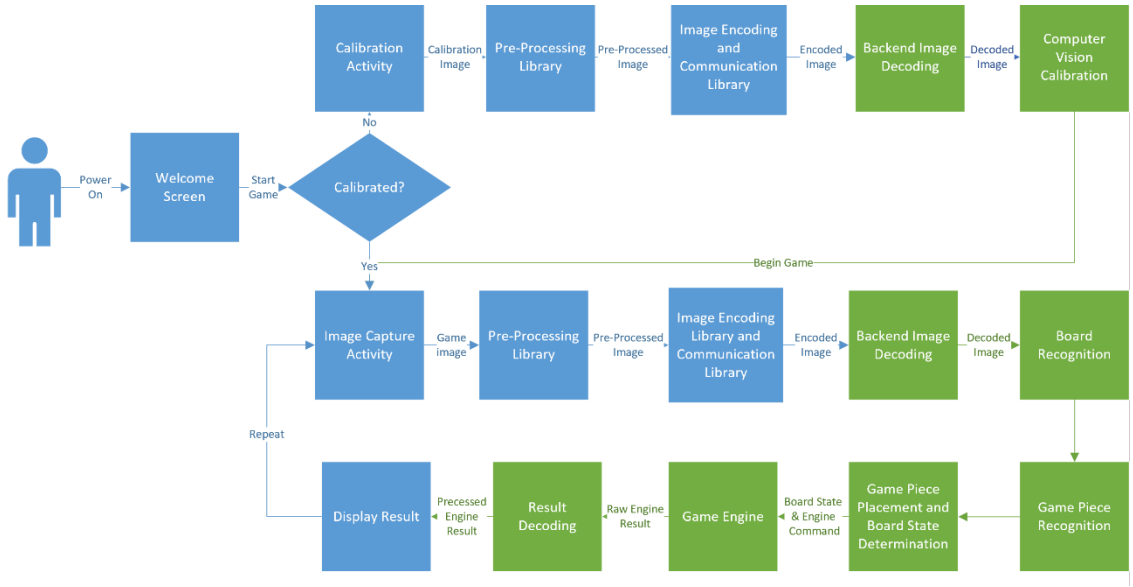

Figure 6: Component Flow Diagram

Currently, we are operating under the assumption that all the components will run on the AR Glasses. As a result, we have no initial pairing or connections to the backend to manage, greatly simplifying the flow. 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 7.

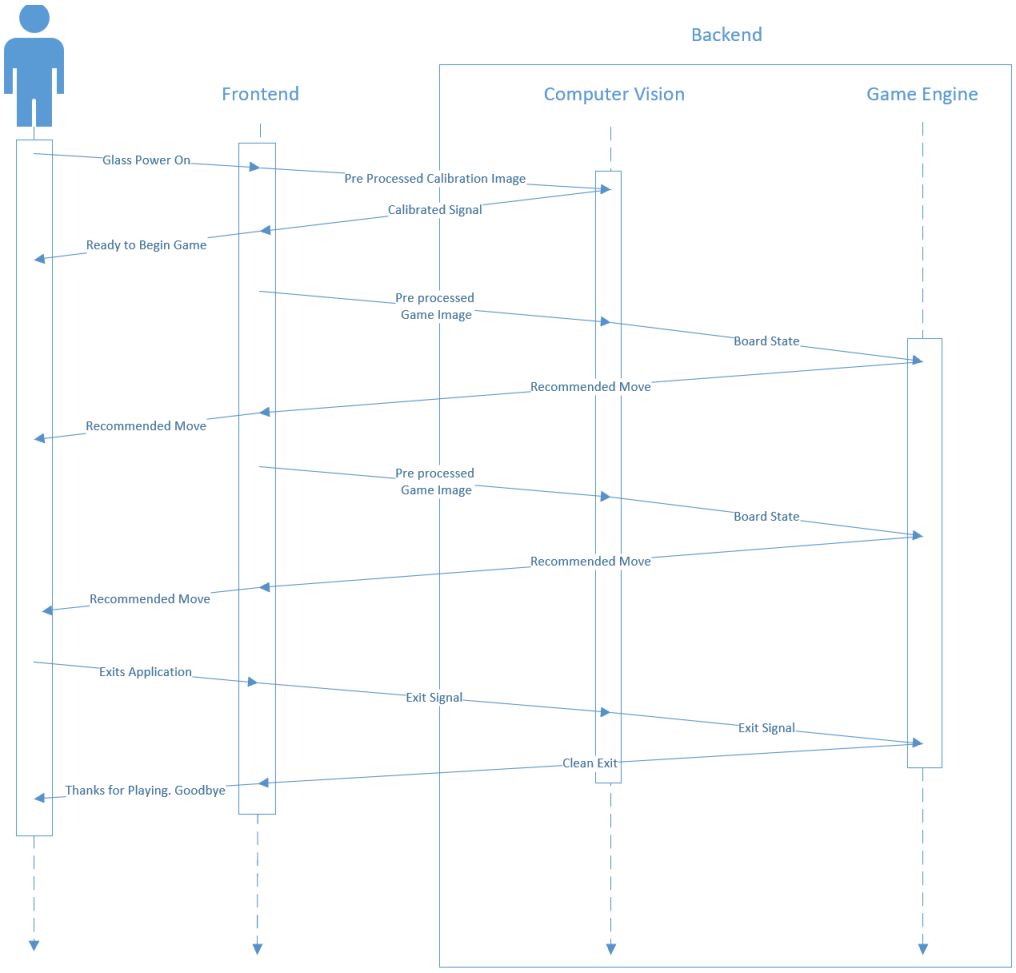

Figure 7: High Level Sequence Diagram

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 5: 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
- Image Capture Activity
- Calibration Activity
- Initial Setup Activities (Not needed if the backend lives on the Vuzix Blade)
- Display Move Activity

4 Testing

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

Additionally, we will perform regression testing throughout the project. This is especially important to the project's computer vision aspect as we continue to tweak algorithms to determine the best performance. For front-end and backend modules, these regression tests will be less frequent and coincide at a minimum for each of the major releases (Prototype, Minimum Viable Product, and Final Design) but will ideally be done for each demo.

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

4.1 Unit Testing

We will conduct individual unit tests on two major Android components. The first is the libraries for communication. We will test each of these libraries and their functions individually to ensure that they perform their specific functions. These communication libraries are crucial to our project's overall functionality and are used by almost every screen we develop; therefore, we need to make sure the base is stable before we start building on top of it. Additionally, communication libraries, when combined with other features, can be very difficult to debug. The communication library testing will be fully automated, and the library should pass all tests to be considered complete.

The final Android software unit we will do unit tests for is image pre-processing. We will test to make sure that the image processing functions perform their intended functions by passing dummy images to a function and verifying the expected output. The visual output of these tests will need to be verified by a human, and our testing lead Brett, will sign off on this unit.

Another software unit that will be tested individually is the game engine. Testing this module will be done by sending commands to the engine and looking for an expecting response and should ensure that the output we are getting from the engine is a viable move for the user to make. The library for sending commands will be tested by a human, ensuring the resulting output of the command is as expected, while the tests of the library for decoding the output will be automated, and the unit should pass all tests to be considered complete.

In order to test our computer vision implementation, we will implement regression testing on a series of test images. The test images will be taken via the Vuzix Blade device and will contain a variety of known game board states and orientations. Finally, we will compare the computer vision pipeline's testing results with the known correct states, and we will require a certain percentage of accuracy in our final product. The computer vision implementation unit testing will be fully automated.

4.2 Interface Testing

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

- Welcome Activity: This activity is shown when the user powers on the glasses. If it becomes necessary to host a separate backend application, this activity will also be responsible for establishing the appropriate connections such as WIFI or potentially Bluetooth to a paired device.
- Initial Setup: This activity would be displayed if the backend needs to run on a separate device. This activity aims to display the setup information for pairing with this device, initiate pairing requests, and then save the paired device such that it can be connected to automatically on power on in the future.
- Calibration: This activity is used to capture and send the necessary calibration images to the computer vision module, wherever it ends up living.
- Image Capture: This activity is used to capture images, do pre-processing, and send the images to the computer vision module, wherever it ends up living.
- Displaying a Recommended Move: This activity will display the decoded move recommended from the game engine in an understandable text-based format for the user.
- Customizing the Game Engine: This activity will allow the user to customize the game engine's difficulty and timing, such that if they are playing under time constraints, the engine gives the user a recommended move within the user's requirements.

Additionally, the computer vision interface will need to undergo testing. In addition to our regression testing (described above in our unit testing), we will test our communication between various portions of the front-end and backend implementations. These interfaces include:

- Image Receiving: This interface allows the Android application running on the AR device to communicate an image from the Android image capture interface to the computer vision pipeline.
- Game Engine Translation: This interface allows for "translation" from the game board's image into a game state communicable to the game engine.

These interfaces are easily tested and verified by humans and can use the test data generated from the unit testing described above to minimize repetition and duplicate effort. These interfaces will be tested individually and by hand, using mock data when necessary and real data where possible. Our Test Lead, Brett will be responsible for coordinating the testing plan for these activities as well as the acceptance criteria starting in mid-February 2021.

4.3 Acceptance Testing

Acceptance testing for us will consist of two levels. The first level is where we combine two or more of the interfaces together to create a more complete flow through the app. These flows and their description can be found below:

- Welcome and Initial Setup: The user should be able to turn on the glasses and be guided through an initial setup (if necessary) to pain the glasses to a companion mobile device before returning to the welcome screen. After the initial setup has been completed once, the user should only be shown the welcome screen while the device auto connects to the saved device.
- Calibration and Image Capture: The user should be able to capture calibration images and send them to the computer vision module to calibrate the algorithms. After doing so, a signal should be sent back, and normal game board images should be captured for gameplay.
- Game Engine Customization and Displaying Result: The user should be able to customize the engine to meet their strength and time requirements, and we should be able to see faster response times to the display.
- 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 tests cases are used to test each of the individual functional requirements (1.4) amongst the development team. These level one tests will be tested individually and by hand, using mock data when necessary and real data where possible. Our Test Lead, Brett, will be responsible for coordinating the testing plan for these activities using the functional requirements to determine the acceptance criteria starting in mid-February 2021.

Upon passing the level one tests among our team members, we will integrate all components into our final design and begin testing internally by playing full chess games. After we feel confident that our design is working as expected, we will take our design to our Product Owner and client Dr. Zambreno so that he can play some games and provide feedback, especially regarding any non-functional requirements he feels should be added. We will iterate on the design based on his suggestions and then present our iterated design to him again. At this time, we would also look to test the product on other chess players. This could include people at the library, members of clubs such as IEEE, Critical Tinkers, the ISU Chess Club, or anyone interested. We will compile the results and feedback provided to us and iterate on the product once again before showcasing our final product to Dr. Zambreno. We hope that through our user tests, we can refine our prototype to address user pain points to allow for more straightforward use.

4.4 Results

So far, the only testing we have done has been eye tests for both the front-end and the backend, as well as encoding and decoding tests. On the front-end, we have developed only two screens so far that we have tested on an Android phone to verify that they do indeed work. The other testing we have done on the front-end is unit testing the JSON formats and image and text encoding and decoding, all of which have succeeded.

The only backend testing we have done so far has been eyeball tests on the algorithm. In doing these tests, we found the methods and algorithm was not as viable as we thought, and we pivoted

to another method in mid-October. The initial approach struggled with efficiently detecting the board's lines and determining the squares, something that is essential for our project. Therefore, we pivoted to a new method that is producing better results to the eye, and we plan to move forward with that approach.

Below, in Figures 8 – 11, are the results of our computer vision algorithm testing so far:

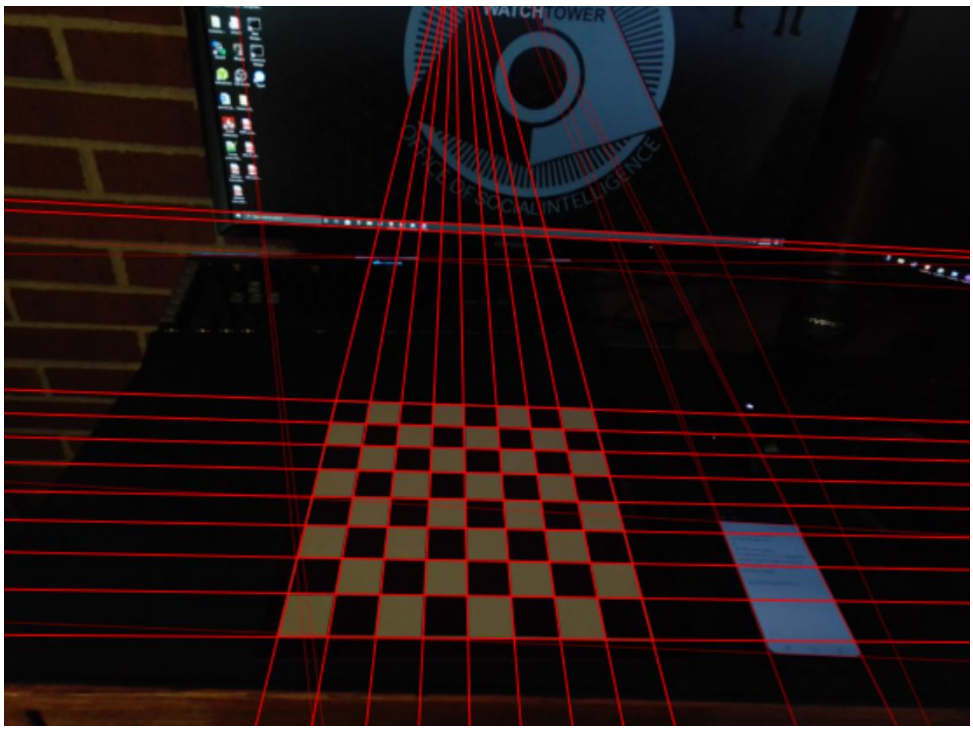

Figure 8: Line Detection

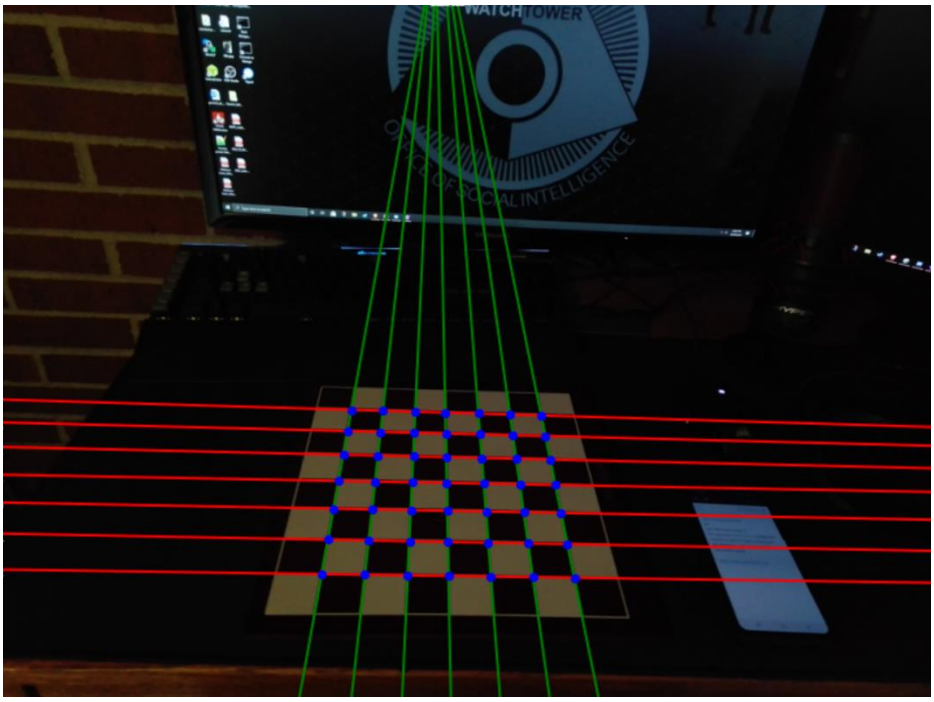

Figure 9: Internal Point Detection

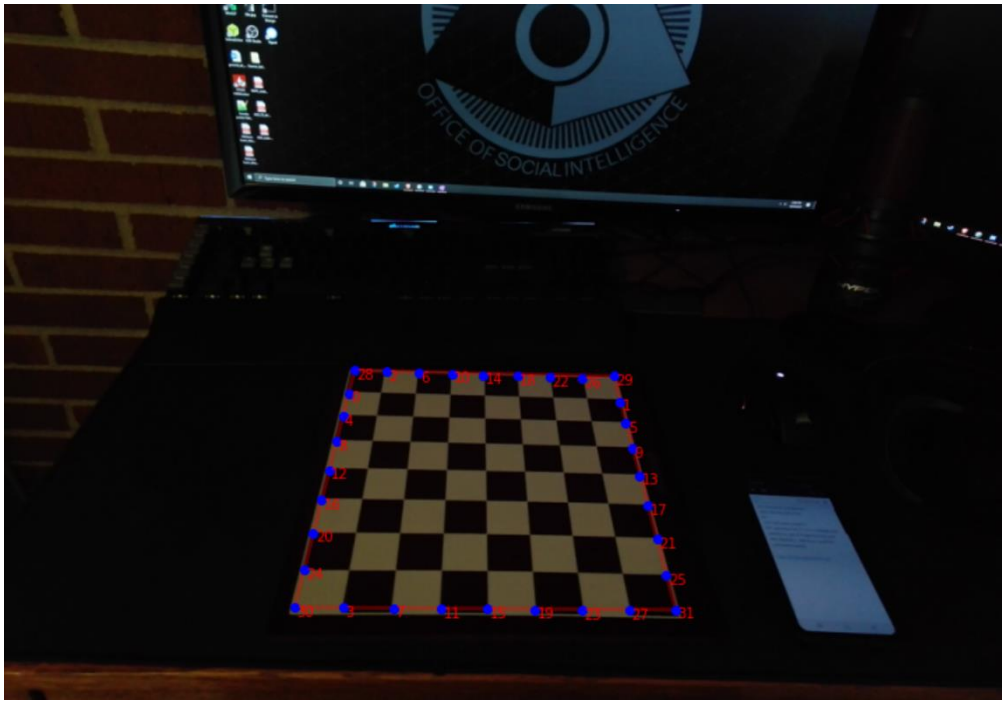

Figure 10: External Point Detection

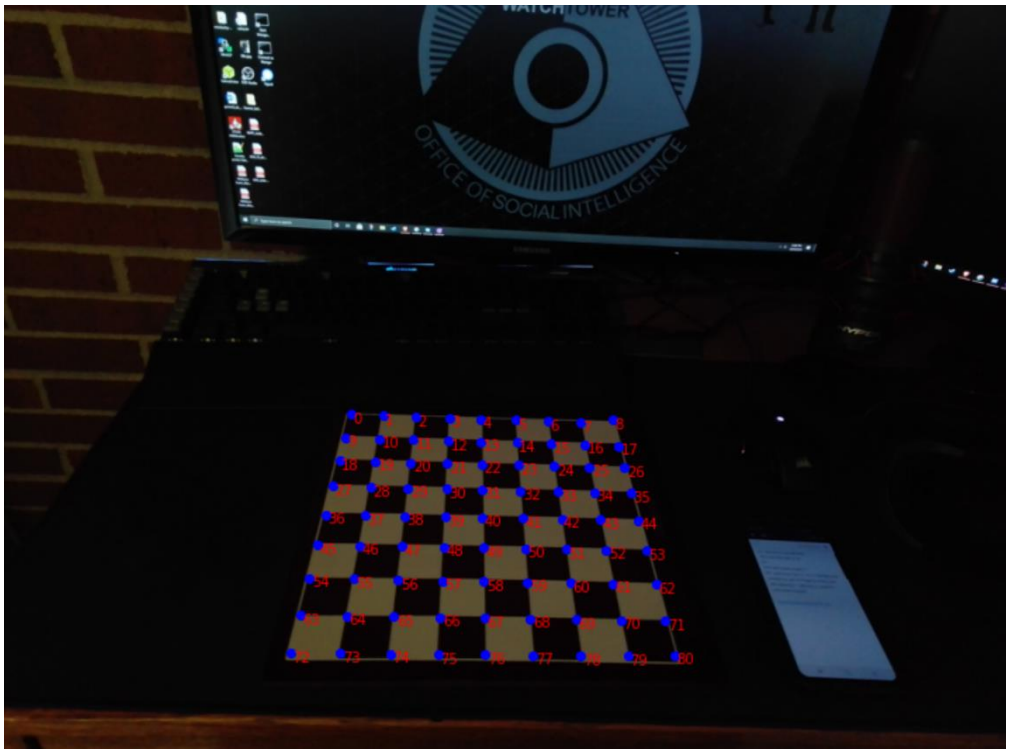

Figure 11: Full Point Detection

As you can see from the images above, the computer vision module of the project is coming along nicely can currently can recognize and detected the board correctly, accomplishing the first of three elements of this module required to complete the project.

5 Implementation

Our team started development in early October according to the tasks on the Gantt chart. So far, the team has been extremely successful in both the front-end and backend areas of the project. On the front-end side, the team has the communication and encoding/decoding libraries, pre-processing image libraries, as well as two key screens. The first screen is the Welcome Activity shown below in Figures 12 and 13.

Figure 12: Game Advisor Welcome Screen

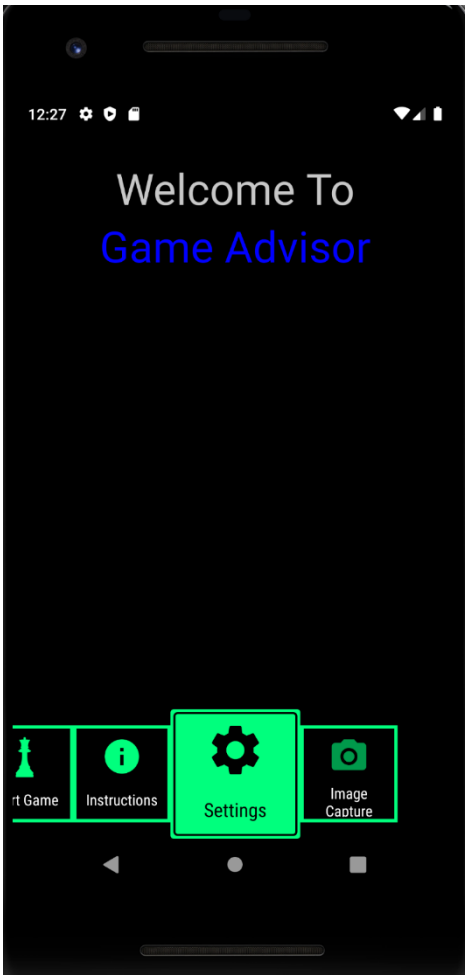

Figure 13: Game Advisor Welcome Screen Scrolled Right

Figures 12 and 13 show the Vuzix Blade application running on Google Pixel emulator, so the size is not to scale, however the features are all still present. On the actual device, the application will not have all the blank space in the middle between the tittle and the scroll bar at the bottom. This scroll bar is Vuzix's standard way of navigation within the app. Using the buttons on the side of the device, users can scroll through this menu and select the tile they want. Currently only the Start Game and Image Capture Tiles are hooked up while the others will show Toasts to signify that it would open another page.

Below in Figure 14, you seen the output of the Image Capture Activity when running the application on the Vuzix Blade device. Note that this screen 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 if very 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 an camera preview so that you can see what the picture would be off before you capture the image.

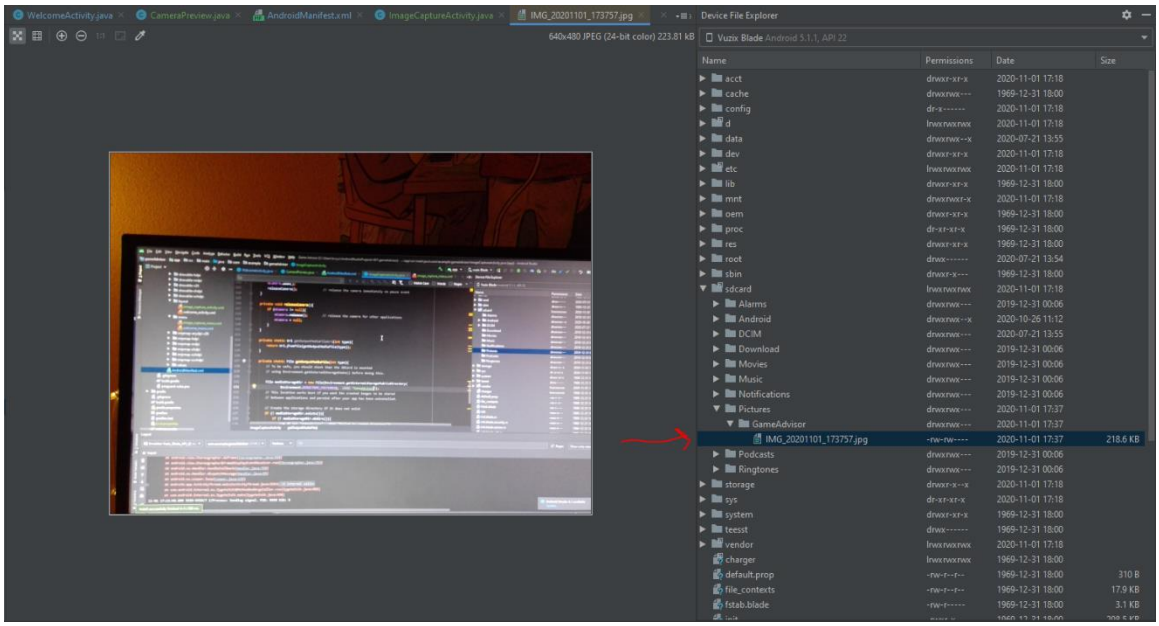

Figure 14: Proof of Image Capture

Currently, we do not have the computer vision pipeline imported into the Android project, and therefore, we have nothing to pass the image to once taken. Currently, we are just saving the image to the device's SD card to show that the image was in fact, captured.

Figure 15, below, shows an example recommended move that would appear after an image has been captured and processed. Note this screen, like figures 12 and 13 is emulated and not to scale.

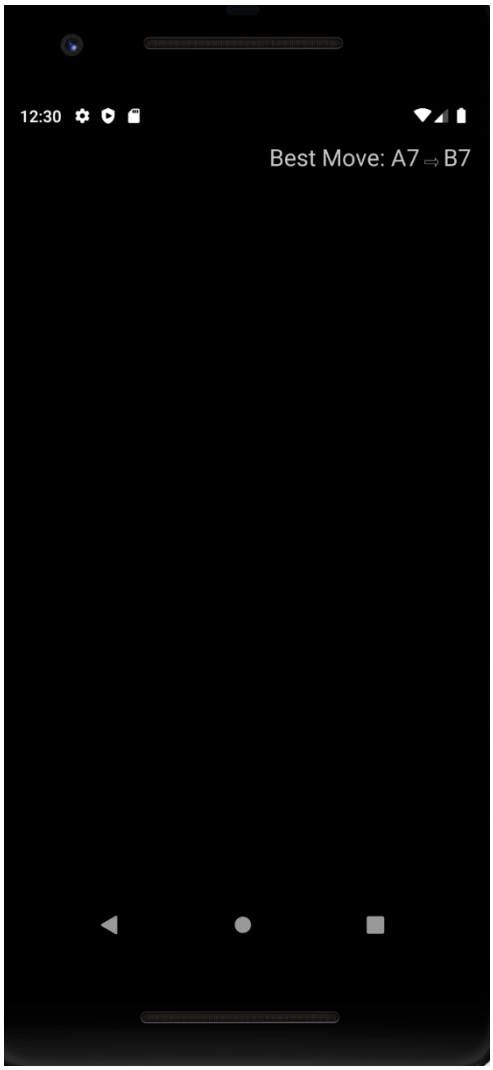

Figure 15: Example Recommended Move

This text will appear in standard chess notation on the top right of the image capture screen, allowing for the user to continue to repeat the loop of capturing images and getting recommended moves without navigation on their part.

Ideally, these screenshots of the application would have been taken on the actual Vuzix Blade device, however, attempts to take pictures of the display on the glasses were unreadable and blurry. We hope to be able to demo these screens using the actual device next semester assuming we have the opportunity to do so in person.

On the backend side of things, we have been able to get the Stockfish Engine binary imported into the application and have written libraries for sending commands and decoding the output. The computer vision pipeline is also coming along nicely. At this point in the project, we can detect the board state very well, as you can see in some of our testing/implementation images in Figures 8-11 above. We have not yet started on chess piece detection as we had initially planned, however, we feel we are in a great position to continue implementation in CPRE 492 in January.

Our implementation plan for CPRE 492 picks up right where we left off in CPRE 491. Our current Gantt Chart and project plan has initial task development continuing until the end of February. The plan does not include any revisions, modifications, or iterations that we undertake as a result of our prototype demo at the end of November. During these first two months of 2021, the front-end development work is a bit light in comparison to the computer vision work that is still being finished. As a result, at least one front end developer will begin iterating on the existing screens based on the feedback from our prototype demo. Depending on the iterations required at this point, we may also have a developer or two start working on nice to have features such as visual overlays rather than just text. We will also be doing user and interface testing during this time.

Once the initial computer vision tasks have been completed and integrated in late February, we will begin our first round of acceptance testing internally as well as the computer vision interface tests. We will iterate once on the results of these tests and then demo the MVP to our Product owner Dr. Zambreno. Taking his feedback, we will go through a series of iterations and further testing to refine our design and implementation. During this time, we will also be working on nice to have features such as AR overlay or expanding the project to support other strategy board games. At this point, we are unsure of where next semester will lead and plan to let the feedback from our demos dictate that path.

Regardless of the path, we plan to implement stricter review standards for next semester. Currently, we are only reviewing merge commits, and this review is being done by only one person, typically the team lead. Since the focus of next semester is on implementation and refining our design, we plan to commit additional time and resources to the review process. This includes implementing many of the processes described in the Standard for Software Reviews and Audits [5]. Specifically, we will be expanding our technical review process to include at least one review approval per traditional commit and at least two reviews for each merge commit. This is in addition to the code passing the CI/CD pipeline containing a variety of tests.

As the semester comes to a close, we will wrap up our implementation and design a week or two early and begin focusing on organizing project material so that it can be passed of to the next senior design group. This includes passing off our shared google drive and source code.

6 Closing Material

6.1 Conclusion

As a team, we have 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 sperate 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.

As we progress through CPRE 492, we plan to have a much more iterative approach to development and design than we have had in CPRE 491. The iteration we will do in this class includes fixing pain points, adding nice to have features, and making changes based on testing failures. This will allow us to best progress towards a final product that our Product Owner Dr. Zambreno and other product users are happy with.

6.2 References

List technical references and related work / market survey references. Do professional citation style (ex. IEEE).

[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.