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: 10/4/2020 Version 1

Executive Summary

Tabletop 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 state of the game board 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)
- Detect 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 tabletop 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 tabletop 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 was not taught in courses

List all new skills/knowledge that your team acquired which was not part of your Iowa State curriculum in order to complete this project.

//TODO

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 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

Tabletop 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 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 state of the game board 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)
- Detect 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 tabletop 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 tabletop 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 will 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:

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 tabletop 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 – mid-September 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 mobile companion. The four pieces of the project may not all work together yet to play the tabletop 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 tabletop 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 tabletop 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 tabletop 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 Backends
        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
        2.3.2.2. Train Classifiers (Haar Cascades; processing can be done in background)
            2.3.2.2.1. Depends on 2.3.1.2
    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
        2.3.3.3. Train Piece Identification Models (like 2.3.2.2, processing can be done in the background)
            2.3.3.3.1. Depends on 2.3.2.2, 2.3.3.2
    2.3.4. Board State Determination
        2.3.4.1. Ensure we can do chessboard layout detection
            2.3.4.1.1. Do line detection
                2.3.4.1.1.1. Depends on 2.3.3.1
            2.3.4.1.2. Check for chessboard layout
                2.3.4.1.2.1. Depends on 2.3.4.1
            2.3.4.1.3. Only highlight chessboard
                2.3.4.1.3.1. Depends on 2.3.4.2
        2.3.4.2. Ensure we can find all the chess pieces

2.3.4.2.1. Ensure we can find a single chess piece
2.3.4.2.1.1. Depends on 2.3.2.2
2.3.4.2.2. Extend to find multiple chess pieces
2.3.4.2.2.1. Depends on 3.3.4.4
2.3.4.3. Detect the type of each chess piece found
2.3.4.3.1. Depends on 2.3.4.4
2.3.4.4. Detect the position of each chess piece on the board.
2.3.4.4.1. Depends on 2.3.4.4, 2.3.4.5
2.3.4.5. 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 Backend. 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
- 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 chess board 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.
- 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 out protype 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 AR Glasses Ordered no later than 9/18/20

- Begin Proof of Concept Development no later than Oct 1, 2020
- Computer Vision Algorithms can detect chessboard and recognize a few pieces no later than the Prototype Demo at the end of the Fall Semester
- Proof of Concept Development Complete by the end of the 2020 fall semester
- Minimum Viable Product Complete by mid-March 2021
- Finalized Design Complete by the end of Spring 2021

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. Table 1 below matches each task up to its respective metrics.

Task Number	Top Level Module	Task Name	Subtask Description (If Needed)	Further Subtask Description (If Needed)	Task Metrics
1	Generate Requirements				
1.1	[redacted]	Meet with Product Owner and Advisor Dr. Zambreno for project kickoff			N/A
1.2	[redacted]	Research the problem and empathize with users			Number of areas researched, types of users explored, use cases determined
2	Market Research				
2.1	[redacted]	AR Glasses			Number of viable options found and explored
2.2	[redacted]	Computer Vision			Number of viable options found and explored
2.3	[redacted]	Chess Engine			Number of viable options found and explored
2.4	[redacted]	Network Communication and Protocols			Number of viable options found and explored
2.5	[redacted]	Existing			Number of viable

	⬛	Mobile Backends			options found and explored
3	Define Solution				
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			
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				
1.1	⬛	Project Creation			
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			
1.2.1	⬛	🟩	Create Welcome Activity to be seen upon Glass Power On		% Complete? # of Screens completed? Communication Calls hooked up?
1.2.2	⬛	🟩	Connect to saved paired devices. This many not be necessary if the		Can we connect to one device? How many devices are saved that

	⬛	🟩	entire app can run on the Vuzix Blade		we try to connect to?
1.3	⬛	Initial Setup			
1.3.1	⬛	🟩	Develop Activities 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 the entire app can run on the Vuzix Blade		Can we save a device? # of devices we save?
1.4	⬛	Calibration			
1.4.1	⬛	🟩	Develop Activities needed for camera and computer vision algorithm calibration		% Complete? # of Screens Completed? Communication Calls hooked up? Able to send and receive things with the computer vision module? (Y/n) What are we able to send and receive currently? (Text, Images)
1.5	⬛	Image Capture			
1.5.1	⬛	🟩	Develop Activities needed for capturing an image		% Complete? # of 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 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			
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			
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			
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.		% complete? What stage are you in? (research, import, customize, or develop) Portability? Easy for 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				
2.1	⬛	Project Creation			
2.1.1	⬛	🟩	Create the project within Android Studio and select the Design Layout and .gitignore		Completed? (Y/n)
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.		% 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			
2.3.1	⬛	🟩	Initial Setup		
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		
2.3.2.1	⬛	🟩	🟦	Ensure the images we receive are in a standardized format (file type, size, orientation, etc.)	Completed? (Y/n)
2.3.2.2	⬛	🟩	🟦	Train Classifiers (Haar Cascades; processing can be done in background)	% complete? What stage are you in? (research, import, customize, or develop) Portability? Easy for other activities to call methods in a library?
2.3.3	⬛	🟩	Image Processing		
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?
2.3.3.3	⬛	🟩	🟦	Train Piece Identification Models (like 2.3.2.2, processing can be done in the background)	% 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		
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	% complete? What

	⬛	🟩	🟦	multiple chess pieces and place accurate bounding boxes around them	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			
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)	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		
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		
2.4.3.1	⬛	🟩	🟦	Develop, import, or customize 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 1, below, contains the milestones discussed in section 2.3.

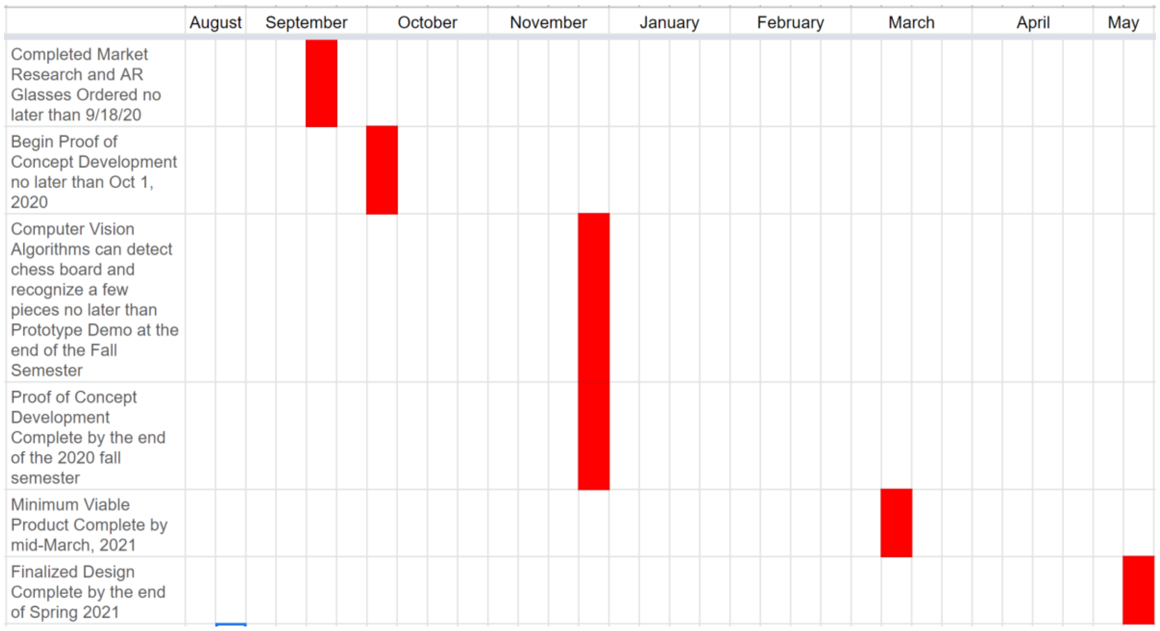

Figure 1: Gantt Chart – Milestones

Figure 2, 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.

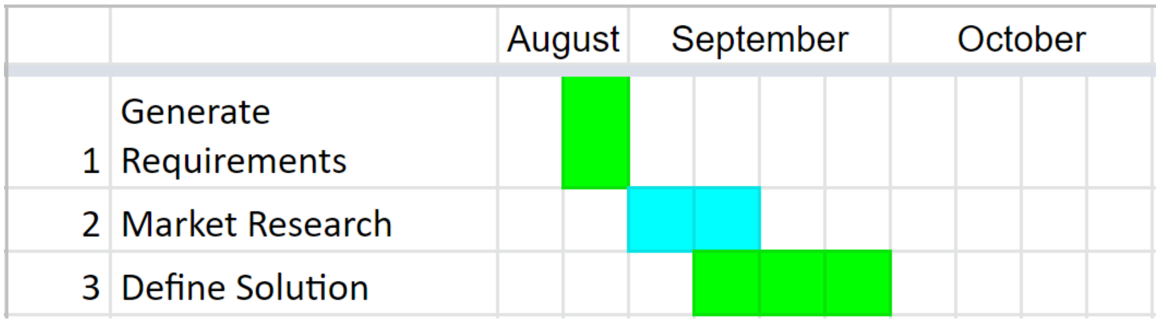

Figure 2: Gantt Chart - Project Planning and Design

Figure 3, 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 significantly. 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.

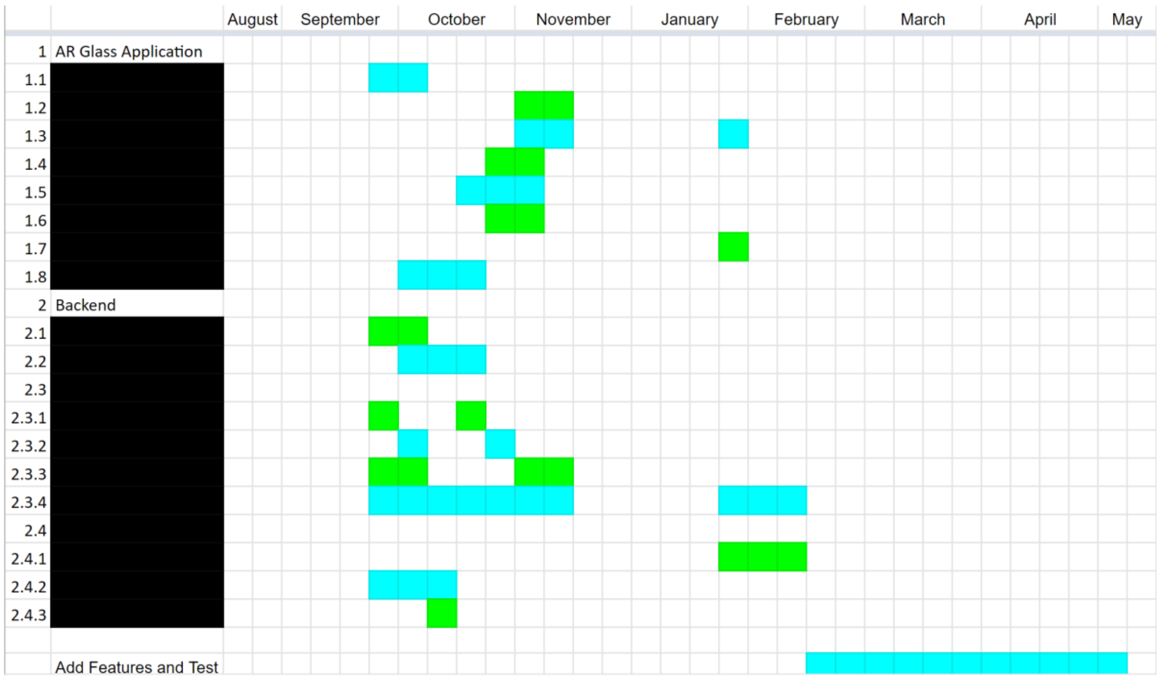

Figure 3: Gantt Chart - Development

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. For important discussions, our team has two weekly meetings: 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 tract tasks than google drive.

2.6 Personnel Effort Requirements

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.

Task Number	Top Level Module	Task Name	Subtask Description (If Needed)	Further Subtask Description (If Needed)	Total Effort (Hours)
1	Generate Requirements				
1.1	[redacted]	Meet with Product Owner and Advisor Dr. Zambreno for project kickoff			7
1.2	[redacted]	Research the problem and empathize with users			13
2	Market Research				
2.1	[redacted]	AR Glasses			12
2.2	[redacted]	Computer Vision			12
2.3	[redacted]	Chess Engine			12

2.4	⬛	Network Communication and Protocols			12
2.5	⬛	Existing Mobile Backends			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 an 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),		3

	⬛	🟩	Includes implementing library calls into activity		
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 the Design Layout and .gitignore		3
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		3

	⬛	🟩	decoding images passed from the front end		
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
2.3.2.2	⬛	🟩	🟦	Train Classifiers (Haar Cascades; processing can be done in background)	20
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
2.3.3.3	⬛	🟩	🟦	Train Piece Identification Models (like 2.3.2.2, processing can be done in the background)	20
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 boxes 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 Backend 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, 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

Include relevant background/literature review for the project

– If similar products exist in the market, describe what has already been done

– If you are following previous work, cite that and discuss the **advantages/shortcomings**

– Note that while you are not expected to "compete" with other existing products / research groups, you should be able to differentiate your project from what is available

Detail any similar products or research done on this topic previously. Please cite your sources and include them in your references. All figures must be captioned and referenced in your text.

Our project is similar to the 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 [//TODO CITE THIS]. 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.

//TODO Computer Vision Articles

3.2 Design Thinking

~~Detail any design thinking driven design "define" aspects that shape your design. Enumerate some of the other design choices that came up in your design thinking "ideate" phase.~~

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 the initial problem to be more general and include all tabletop games to allow for flexibility. 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 Halo Lens. In conjunction with the AR Glass ideation, we looked into a variety of computer vision techniques and libraries including OpenCV, Computer Vision Toolbox, ML.Net, and Google Colab.

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 if it would prove to difficult we wanted to have other simpler options. 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 and 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 flushing out a prototype development plan.

3.3 Proposed Design

~~Include any/all possible methods of approach to solving the problem:~~

- ~~Discuss what you have done so far – what have you tried/implemented/tested?~~
- ~~Some discussion of how this design satisfies the **functional and non-functional requirements** of the project.~~
- ~~If any **standards** are relevant to your project (e.g. IEEE standards, NIST standards) discuss the applicability of those standards here~~
- ~~This design description should be in **sufficient detail** that another team of engineers can look through it and implement it.~~

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 chess board 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 snapshot of the chess board using AR Glasses
5. Send snapshot to the computer vision module
6. Run computer vision algorithm to determine the chess board state
7. Pass the board state to the game engine to determine the recommended move
8. Send recommended move the 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 of 3 modules of our project is broken down below.

AR Glasss Application:

- Using the Vuzix Blade's camera, the application will take a picture of the chess board and pass this image to the paired mobile device.
- The camera on the Vuzix will need to be calibrated upon first time use using standard OpenCV calibration algorithms.
- In order to share the image to 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 chess board, 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 chess board.
- Setup algorithms to ensure camera calibration.
- We will likely use trained image classifier to locate and identify chess pieces on the chess board.
- Some image preprocessing 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 be responsible to 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 pack 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

Highlight the strengths, weakness, and trade-offs made in technology available.

Discuss possible solutions and design alternatives

When analyzing technology, we looked at three areas: AR Glasses, Computer Vision, andChess Engine. We found a variety of AR Glass options, however only 3 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 Halo Lens (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.

//TODO Computer Vision

We considered a variety of chess engines such as StockFish, Komodoo, Cuckoo, Leena Chess Zero, and Shredder. These engines all have similar capability 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

– Did your proposed design from 3.3 work? Why or why not?

– What are your observations, thoughts, and ideas to modify or iterate over the design?

3.6 Development Process

~~Discuss what development process you are following with a rationale for it – Waterfall, TDD, Agile. Note that this is not necessarily only for software projects. Development processes are applicable for all design projects.~~

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 best manage our time 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 weeks of initial setup, we will be we positioned to transition to 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 best meet the client's needs.

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 having a greater focus on individual statuses, integrations, and demos.

3.7 Design Plan

Describe a design plan with respect to use-cases within the context of requirements, modules in your design (dependency/concurrency of modules through a module diagram, interfaces, architectural overview), module constraints tied to requirements.

4 Testing

Testing is an **extremely** important component of most projects, whether it involves a circuit, a process, or software.

1. Define the needed types of tests (unit testing for modules, integrity testing for interfaces, user-study or acceptance testing for functional and non-functional requirements).
2. Define/identify the individual items/units and interfaces to be tested.
3. Define, design, and develop the actual test cases.
4. Determine the anticipated test results for each test case
5. Perform the actual tests.
6. Evaluate the actual test results.
7. Make the necessary changes to the product being tested
8. Perform any necessary retesting
9. Document the entire testing process and its results

Include Functional and Non-Functional Testing, Modeling and Simulations, challenges you have determined.

//User Testing

4.1 Unit Testing

– Discuss any hardware/software units being tested in isolation

4.2 Interface Testing

– Discuss how the composition of two or more units (interfaces) are to be tested. Enumerate all the relevant interfaces in your design.

4.3 Acceptance Testing

How will you demonstrate that the design requirements, both functional and non-functional are being met? How would you involve your client in the acceptance testing?

4.4 Results

– List and explain any and all results obtained so far during the testing phase

- Include failures and successes
- Explain what you learned and how you are planning to change the design iteratively as you progress with your project
- If you are including figures, please include captions and cite it in the text

5 Implementation

Describe any (preliminary) implementation plan for the next semester for your proposed design in 3.3.

6 Closing Material

6.1 Conclusion

Summarize the work you have done so far. Briefly re-iterate your goals. Then, re-iterate the best plan of action (or solution) to achieving your goals and indicate why this surpasses all other possible solutions tested.

6.2 References

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

6.3 Appendices

Any additional information that would be helpful to the evaluation of your design document.

If you have any large graphs, tables, or similar data that does not directly pertain to the problem but helps support it, include it here. This would also be a good area to include hardware/software manuals used. May include CAD files, circuit schematics, layout etc. PCB testing issues etc., Software bugs etc.