Design Document
4ZP6: DieSpy

Team #9

Jackson Cassidy
Wyatt Habinski
Christian Majid
Paul Puscas

January 24, 2025

Revision History

Date	Version	Notes
January 26, 2025	0	
April 3, 2025	1	Updated component diagram and behaviours

Purpose Statement

The purpose of this project is to develop an android application that leverages machine learning to automate dice detection and tracking for tabletop gamers. The app will enhance the gaming experience by providing accurate dice roll results, real time statistics, and group collaboration features, allowing players to focus on gameplay rather than tedious, manual record keeping. This document outlines the systems architecture, detailing its key components and their alignment with the project's goals.

Component Diagram (Appendix Figure 1)

The component diagram illustrates the core structure and relationships within our DieSpy android application. It is comprised of frontend screens that provide user interaction functionality, which interact with the backend managers that handle data processing, management, and real time updates. The Firebase Firestore is a cloud database. CameraX and bluetooth low energy are third party components that facilitate camera functionality and local connection. The trained AI model is utilized for advanced dice detection and value recognition. Together, these components create a cohesive, yet modular system.

Furthermore, some connector lines were omitted to keep the diagram clean and concise. Most screens can be accessed from anywhere using the universal navbar. Additionally, many screens and managers access the firebase manager.

Component-Requirement Relationship

System Component	**Requirements Covered**
Login Screen	• P1: Player login to join session, pull previous data from database • Nonfunctional: Simple, username password field obvious. Secure.
Authentication Manager	• P1: Authenticate player with profile, pulls information securely • Nonfunctional: Security, privacy
Settings Screen	• P2: Display the users settings and preferences • Nonfunctional: Laymen, nicely laid out
Profile Screen	• P2: Display the profile info and respective profile actions • Nonfunctional: Simple, nice looking UI
Profile Manager	• P2: Saves information to database file about profile, authenticates with the profile manager • P2: Shows settings player has set • Nonfunctional: Secure

Firebase Manager	• P2: Saves previous game, account, and party histories
Dice Simulation manager	• P2: simulate dice roll option • Nonfunctional: Sleek and easy to visualize
Join Party Screen	• P1: Ability to choose party and pulls information from LAN parties • Nonfunctional: Nice feel, efficient, nice layout, speed
Dice Stats Manager	• P2: Calculator roll stats • Saves to roll log and firebase
Logs Manager	• P2: Keep records of past games in database • P2: Ability to see history of games on any user device (host or guest) • Nonfunctional: Usability, look and feel, readability and searchability
Dice Detection Screen	• P0: Displaying camera and overlay boxes • P0: Displaying stats about rolls (sum, number of each number rolled, etc.) • Nonfunctional: Performance, speed, simple UI, readable, usability
Dice Detection Manager	• P0: Pulling info from the AI models • Nonfunctional: Fast, pulling data as soon as possible,
Chat Manager	• P2: All players (host or guest) see a chat log of all the previous rolls and information • P1: Local Lan connection • Nonfunctional: Look and feel, readable

Components Behaviour

Component	Authentication Manager
Normal Behavior	Provides backend to reading and writing. Authenticates provided credentials, and if valid, allows access to the party screen.
Implementation	Class AuthenticationManager { Int login(string HashedUsername, string HashedPassword); String hash(string s) Void loadPartyScreen()

	}
Undesired Behavior and Handling	Could hash improperly, could not read from database correctly. Will have either checksum or retries to ensure the user input is not the error.

Component	Shared preference Manager
Normal Behavior	Provides global variables across components. Also allows for persistence when app is closed

Implementation	Class SharedPreferenceManager {
	currentUserId
	currentPArtyId
	}

Undesired Behavior and Handling	Persistence isn't controlled. Data is improperly reset and updated

Component	PartyCacheManager
Normal Behavior	Provides caches for party data to limit database read and writes

Implementation	Class PartyCacheManager {
	partyId
	userIds
	turnIndex
	chats
	Logs
	clear()
	}

Undesired Behavior and Handling	Cache is not updated, cleared, or loaded properly

Component	FireStoreManager
Normal Behavior	API to firebase firestore database for modularized read and writes

Implementation	Class FireStoreManger {
	getDocument()
	updateDocument()
	deleteDocument()
	}

Undesired Behavior and Handling	Async calls. Unauthorized calls

Component	PartyManager
Normal Behavior	Provides real time update listeners to database for party / game state data

Implementation	Class PartyManager{
	subscribeToLatestLog()
	subscribeToTurnOrder()
	subscribeToPartyMemebers()
	}

Undesired Behavior and Handling	Could not respond to screen inputs, could lose connection, could read/write to incorrect database address

Component	LogsManager
Normal Behavior	Provides backend for managing logging functions

Implementation	Class LogManager {
	List<string, int> getlogs(string FirebaseAddress);
	Int saveLogs(string FirebaseAddress)
	}

Undesired Behavior and Handling	Could write to improper database address

Component	ChatManager
Normal Behavior	Provides backend for party chat functionality

Implementation	Class ChatManager{
	int sendMessage(string Message, string Username, DateTime timestamp, PartyManager pm);
	Int getMessage(string Message, string Username, DateTime timestamp PartyManager pm)
	}

Undesired Behavior and Handling	Desynced messages among users,time

Component	DiceStatsManager
Normal Behavior	Provides backend for maintaining score

Implementation	Class StatsManager{
	List<string,string> retrieveRecords(String player, String firebaseAddress)
	Int writeRecords(
	}

Undesired Behavior and Handling	Could suffer from desyncs, could retrieve information from wrong address

Component	DiceDetectionManager
Normal Behavior	Pulls information from the AI models and makes it into text form

Implementation	Class DiceDetectionManager() {
	List<MLModel> trainedImageModels
	List<BoundingBox> DetectFaces(image screenshot, MLModel model);
	image TakeScreenshot();
	image normalizeImage(image screenshot);
	}

Undesired Behavior and Handling	Dice not detected. Continuously reading the screen and taking the most common input will be the best way to showcase the correct information.

Component	NetworkManager

Normal Behavior	Manages all network related activities, maintains open read/write ports for communication with the host for real time updates

Implementation	Class{
	Int post(string jsonMessage);
	Int parseRecievedMessage(PartyManager pm,string jsonMessage);
	Int recievePort,
	Int sendPort
	}

Undesired Behavior and Handling	Typical network issues could occur (lost connection, failed handshake, etc), could fail to parse messages or send improper messages (bad json format)

Component	Login Screen
Normal Behavior	Provides interface upon startup of app. Allows users to sign in with A username and password. Pass the username/password to the authenticationmanager.

Implementation	Class LoginScreen{
	Int sendCredentials(AuthenticationManager am, string Username, string Password);
	Button loginbutton(),
	Textfield usernameField()
	Textfield passwordField()
	}

Undesired Behavior and Handling	Could not respond to screen inputs. Will crash handle and fix upon restart of the app.

Component	Profile Screen
Normal Behavior	Provides static interface for profile view.

Implementation	Class ProfileScreen {
	List<String> retrieveProfileInformation(ProfileManager pm)
	Button backButton()
	}

Undesired Behavior and Handling	Displays incorrect data or does not display some data.

Component	Setting Screen
Normal Behavior	Provides interface for settings.

Implementation	Class SettingScreen{
	Slider VolumeSlider()
	Slider DarkModeSlider()
	}

Undesired Behavior and Handling	Could not respond to screen inputs. Settings may get messed up. Will allow for restoration of default settings. Screen freezes will be error handled and the app will be restarted accordingly.

Component	Home Screen
Normal Behavior	Launch screen upon login. Gateway to existing parties, joining new parties, and party creations

Implementation	Class HomeScreen{
	Button joinPartyButton()
	Button createPartyButton()
	Button backButton()
	}

Undesired Behavior and Handling	Could not respond to screen inputs.

Component	Join Party Screen
Normal Behavior	Provides interface to join a party

Implementation	Class JoinPartyScreen{
	Textfield joinField()
	List<String> getLocalParties(NetworkManager nm)
	PartyManager joinParty(NetworkManager nm, string Party)
	List<Button> joinPartyButtons()
	Button BackButton()
	}

Undesired Behavior and Handling	Could not respond to screen inputs. Parties may not show at first load. Will have a refresh button and screen freezes will be error handled and the app will be restarted accordingly.

Component	Create Party Screen
Normal Behavior	Provides interface to create a party

Implementation	Class CreatePartyScreen{
	Textfield NameField()
	Button backButton()
	PartyManager createParty()
	}

Undesired Behavior and Handling	Could not respond to screen inputs.

Component	Party Screen
Normal Behavior	The main screen of the party

Implementation	Class PartyScreen {
	List <String, Color, int, Button> playerDisplays
	List <Button> viewPlayerScreenButtons
	String currentPlayer
	Int rollDice(string currentPlayer);
	Int viewPlayerScreen(PartyManager pm);
	Button backButton();
	}

Undesired Behavior and Handling	Could not react to screen input, could improperly track player rolls with regards to players leaving.

Component	MemberScreen
Normal Behavior	Displays the information about a specific player. Leave party Button too
Implementation	Class MemberScreen{ String Player Int Playerscore Color PlayerColor Button BackButton() Button: LeaveParty()

	}
Undesired Behavior and Handling	Could not react to screen input

Component	Logs Screen
Normal Behavior	Pulls information from logs and history manager and displays it onto the screen in a readable manner.

Implementation	Class LogScreen {
	List<string> getlogs(LogManager lm);
	Button backButton();
	Table historyTable();
	}

Undesired Behavior and Handling	Logs were not recorded, or not updated correctly. The solution is to check a few times what was supposed to be read, and ensure the screen is always up to date with the latest info.

Component	Chat Screen
Normal Behavior	Ability to send and receive messages to and from the party chat

Implementation	Class ChatScreen{
	List<string, string, datetime> messagesList
	Button BackButton()
	}

Undesired Behavior and Handling	Desynced messages among users,time, Could not respond to screen input

Component	Dice Detection Screen
Normal Behavior	Receives information from the Dice Detection manager, and draws the bounding boxes averaged over 10 frames
Implementation	Class DiceDetector { drawBoundingBoxes(DiceDetectionManager dm, List<BoundingBox> bbs) Button backButton() Const int updateFrequency

	}
Undesired Behavior and Handling	Detection boxes do not line up with the information. Dice are incorrectly detected and/or displayed

UI Details

Link to Figma:
https://www.figma.com/proto/ejKuD9j6FYaXvKW9BNnh6n/Capstone-UI-Design?node-id=3-5&p=f&t=1aCJ6xgj7Zo77JKA-1&scaling=scale-down&content-scaling=fixed&page-id=0%3A1&starting-point-node-id=3%3A5

-Also see Appendix if link fails (Figures 7-17)

Item Placement: Items were placed according to standard app design conventions. Navigation buttons along the top and bottom, with a “back button” in the top left. This leaves the center of the page–where the user’s eye is naturally drawn–to be used for that screen’s content (camera view, chat, etc.)

Font: Inter was chosen for its proven legibility and variety of weights. It is sans-serif which improves legibility on small screens where our app will be running, and is not jarringly different from the many other sans-serif fonts used across mobile apps. [Note that this paragraph was written in Inter]

Colour: The colours used in our app are not wholly reflected in our Figma prototype. Our plan is to give the users a few palettes to choose from, and ensure that there are palettes that are legible for the 3 types of colour blindness.

-See Appendix for example templates (Figures 2-5)-

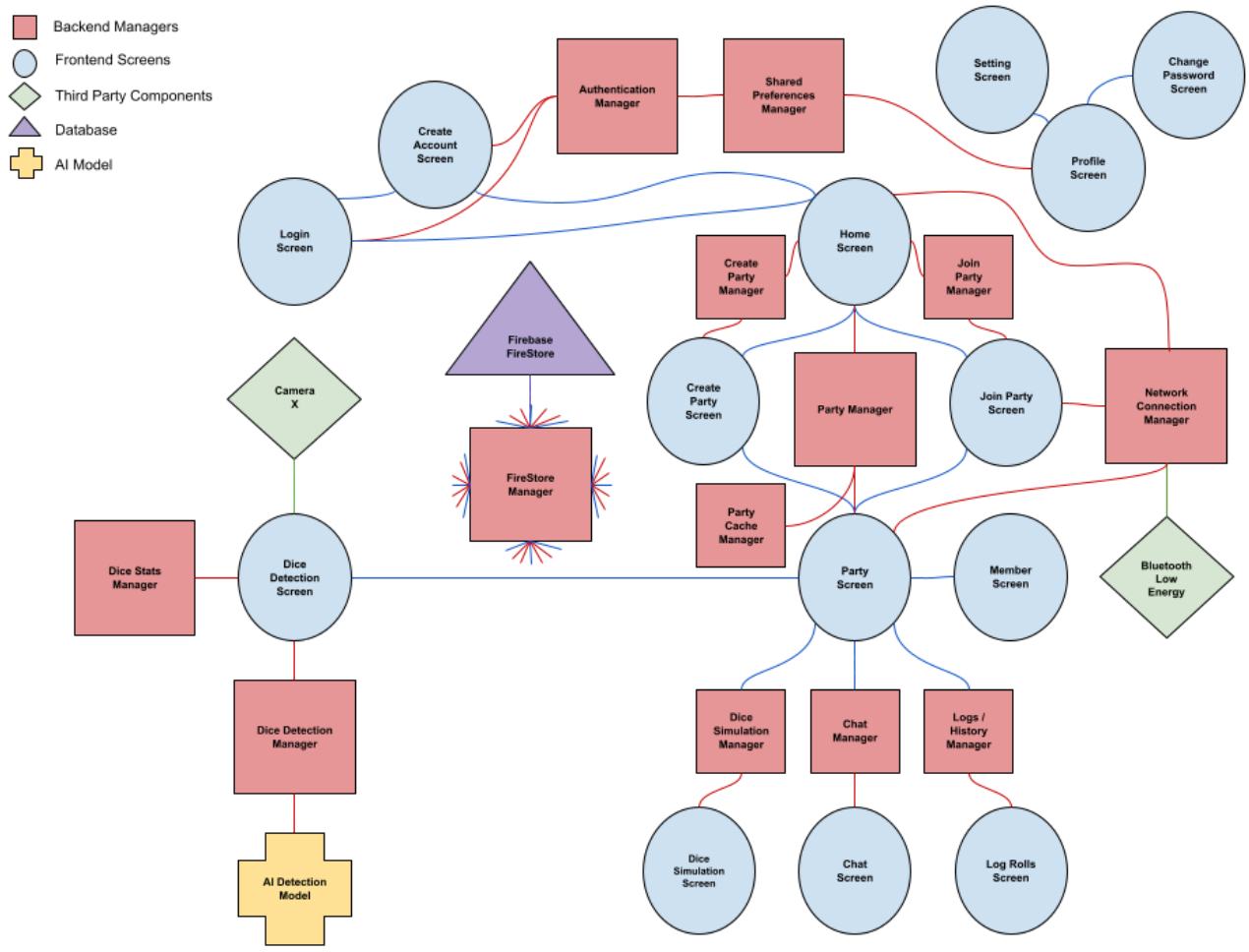
These palettes are based on the colours used in our logo, and their results for colour blindness are in left to right order. Coloring for Colorblindness by David Nichols was used to visualize this.

-See Appendix for example templates (Figures 6-9)-

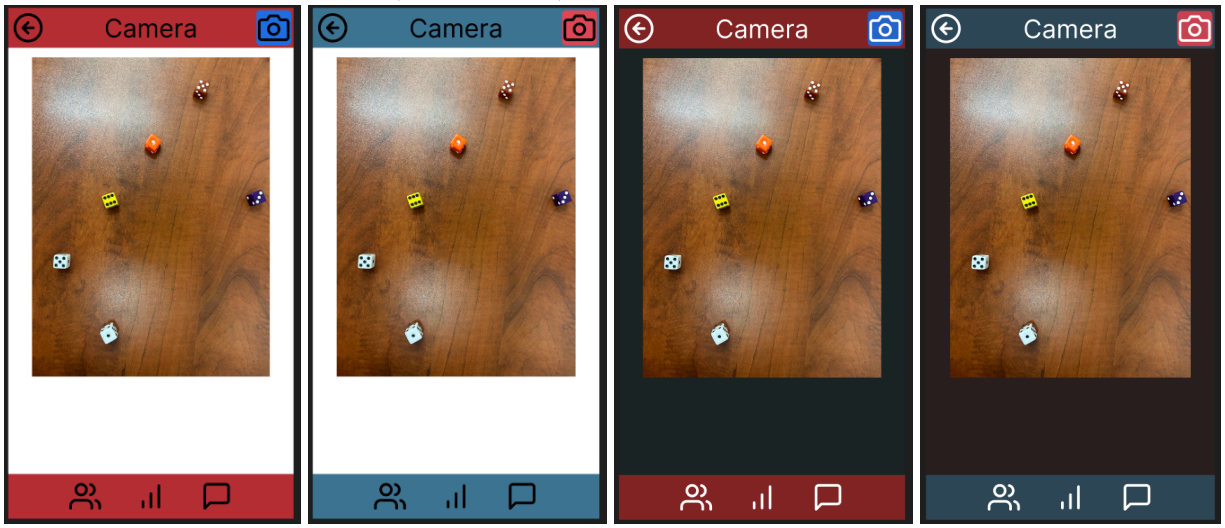
All the colours in each palette appear distinct when adjusted to replicate colour blindness. In the darker palettes the background colour is hard to distinguish from black, therefore white text and icons are used.

Appendix

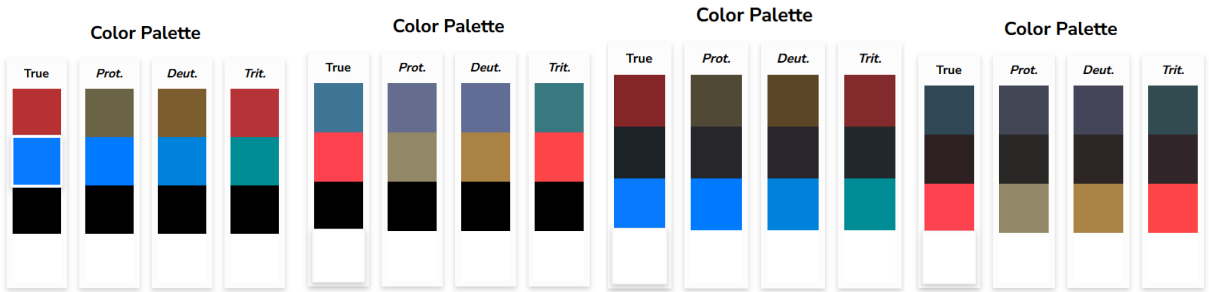
Component Diagram (Figure 1)



Colour Template Examples (Figures 2-5)



Colour Palette Examples (Figures 6-9)



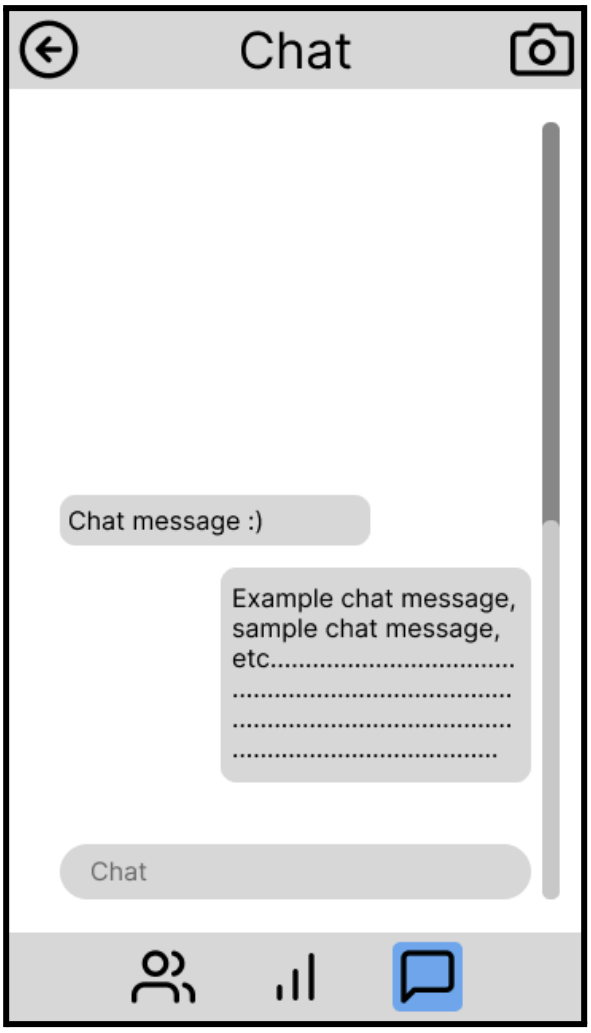
Functional Requirements

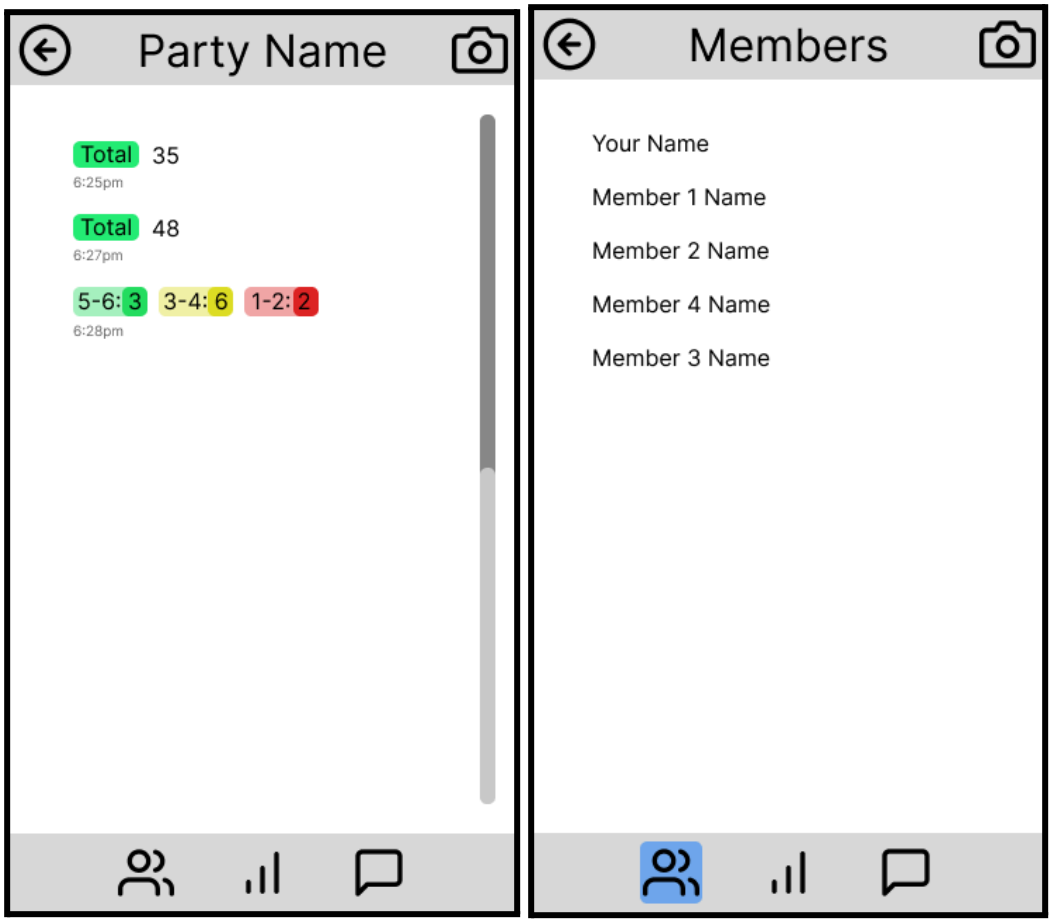
P	Requirement	Details
P0	Camera functionality	The app must be able to obtain camera images of the dice used. The simplest way to do this will be through discrete camera inputs, using CameraX libraries for Android. The intended minimal camera use will consist of a top down photo of the board.
	Dataset	The application must have a fully functional dataset of dice for each model obtained. While this will not be a part of the application per se, it is nevertheless an extremely high priority task. See Data and Metrics for more details.
	Detect and Identify D6s, Count number of D6s, Sum D6	The application must be able to identify as many D6s as are on screen, detect what the value of each die rolled is, and aggregate the results by summing the number of each value rolled.
	Offline mode	By default, the camera and dice detection screen should run without any network connection required.
P1	Scan from Live Video Feed	The app should be able to take a continuous stream of video from the camera and determine which frames to process through the agents. Additionally, the app should not send repeat results after the dice have come to a stop (after a roll but before the dice are picked up).

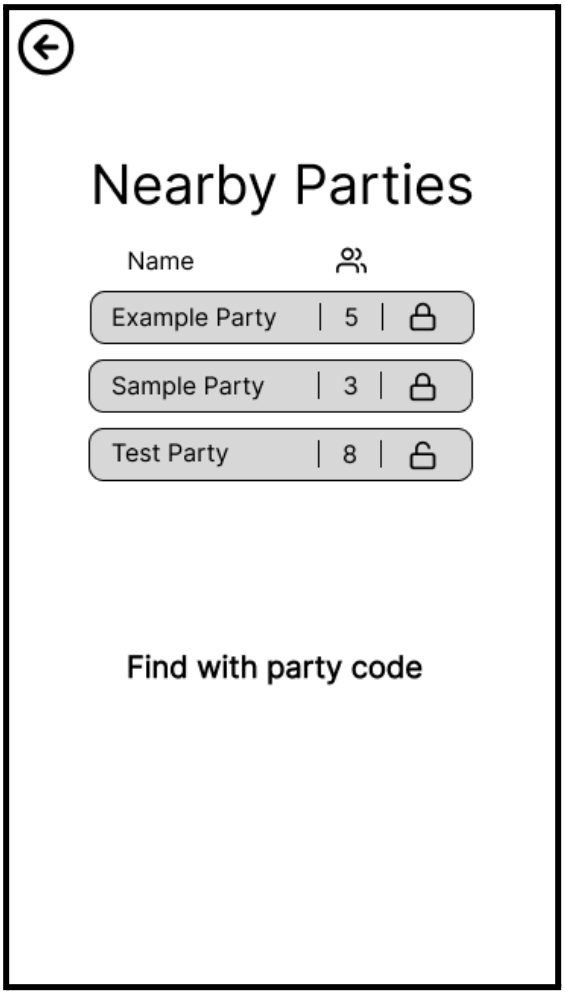
	Login screen, option to Host or Join	The app should have the option to choose between hosting and joining. As well as a screen to connect securely.
	Connect to host and view results	The app should be able to allow hosts and guests to connect using a code, provided they are in physical proximity. Connected guests should have the dice roll results displayed in their app.
	Maintain account/turn system, i.e track rolls by who's turn it is	The app should be able to increment a "turn" through manual input (such as clicking a "Next Turn" button), and should assign the results of a series of rolls to one user.
P2	Log dice roll results	The app should have the ability to show users a history of previous rolls in the current session. This should be transient over games, and should carry over between sessions.
	Save previous game history	The app should keep track of previous players games, and be able to pick up where they left off, or simply go back to view game results
	Editing dice roll (in case of mistake)	The app should give the host the option to approve or deny or change a dice roll result before it is sent to the guests.
	Statistics by account	The app should be able to present aggregate scores to each user as to the rolls that were part of their turns.
	Live Chat	The app should have a built-in chat function, allowing users to send text messages to each other during the session. Live updates should also be sent to the chat.
	Simulate Roll option	The app should have a feature to simulate rolls without having to roll them physically, to save time.
P3	Highlighting dice in camera mode	The app should have a toggleable option to highlight dice in the image using the bounding boxes and results produced by the ML agents.

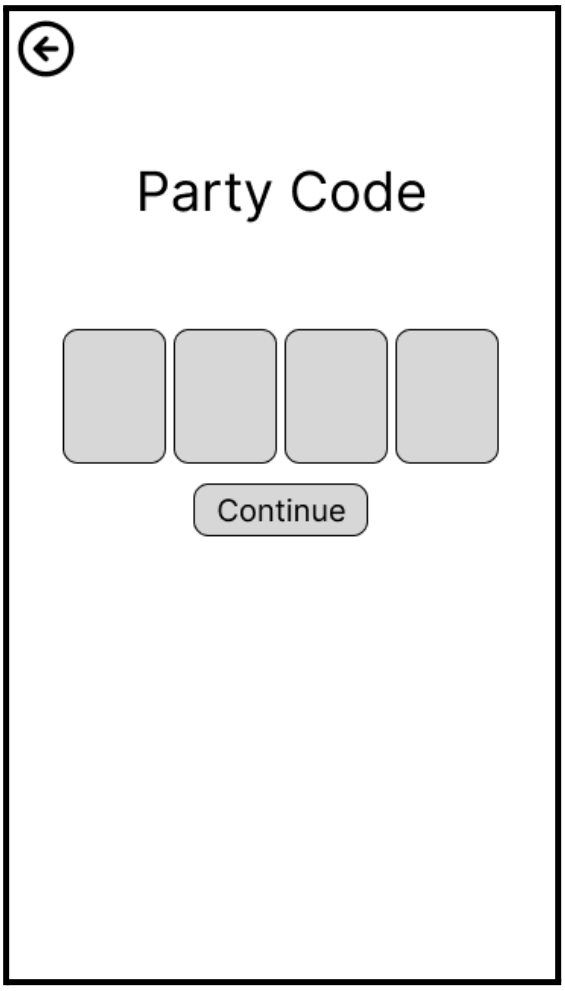
	D4 Support	The app should extend support to recognizing and identifying D4 Dice. This is a technical challenge, as D4s are the only dice with multiple numbers per face.
	Send live footage to guests too	The app should share the live video feed of the host's camera with the guests. The user will be able to toggle between this view and the traditional results view.
	Upload to google play store	The app will be published on the Google Play Store, not only to demonstrate its development quality and robustness, but also to enhance accessibility for users.
	D20, D12, D10, D8 Support	Added support for polyhedral dice beyond the D6. If possible, all supported dice types could be rolled and processed together. Otherwise, the option to switch between dice types will be present.

Figma UI Design (Figures 7-17):




Party Name
Total 35
6:25pm
Total 48
6:27pm
5-6: 3
3-4: 6
1-2: 2
6:28pm
Members
Your Name
Member 1 Name
Member 2 Name
Member 4 Name
Member 3 Name


Nearby Parties
Name
Example Party | 5
Sample Party | 3
Test Party | 8
Find with party code


Party Code
Continue


Create a party
Name
Party Code: 1234
Create


Create/Join A Party
Create
Join
Or
Solo Mode

Stats

Roll 7 6:27pm

Results:

6s: 3 | 5s: 1 | 4s: 2

3s: 5 | 2s: 0 | 1s: 2

Total: 48

Camera