BLIND MATE CHESS

A major project report submitted in partial fulfillment of the requirement for the awarded degree of

Bachelor of Technology in

Computer Science and Engineering

Submitted by

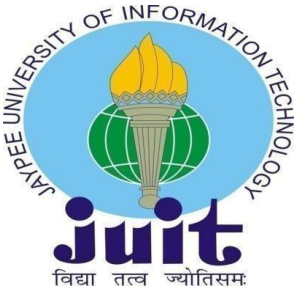
Sheel Bhadra Joshi (201175)

Mohit Verma (201173)

Under the guidance and supervision of

Dr. Ruchi Verma

Assistant Professor (SG)



Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat 173234, Himachal Pradesh, INDIA

Table Of Contents

List Of Figures

List Of Tables

DECLARATION

We hereby declare that the work presented in this report entitled **"Blind Mate Chess"** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science & Engineering** submitted in the Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat is an authentic record of our own work carried out over a period from August 2023 to May 2024 under the supervision of **Dr Ruchi Verma** (Assistant Professor (SG), Department of Computer Science & Engineering and Information Technology).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Mr. Sheel Bhadra Joshi
(201175)

Mr. Mohit Verma
(201173)

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Dr Ruchi Verma
Assistant Professor(SG)
Department of Computer Science & Engineering and Information Technology Jaypee University of Information Technology

CERTIFICATE

This is to certify that the work which is being presented in the project report titled **"Blind Mate Chess"** in partial fulfillment of the requirements for the award of the degree of B.Tech in Computer Science And Engineering and submitted to the Department of Computer Science And Engineering, Jaypee University of Information Technology, Waknaghat is an authentic record of work carried out by **"Sheel Bhadra Joshi (201175)"**, **"Mohit Verma (201173)"** during the period from August 2023 to May 2024 under the supervision of **Dr. Ruchi Verma**, Assistant Professor (SG), Department of Computer Science and Engineering, Jaypee University of Information Technology, Waknaghat.

Submitted by:

Mr. Sheel Bhadra Joshi
(201175)

Mr. Mohit Verma
(201173)

The above statement made is correct to the best of my knowledge.

Dr Ruchi Verma
Assistant Professor(SG)
Department of Computer Science & Engineering and Information Technology Jaypee University of Information Technology

ACKNOWLEDGEMENT

Firstly, we express our heartiest thanks and gratefulness to almighty God for his divine blessings that makes it possible for us to complete the project work successfully.

We are really grateful and wish our profound indebtedness to our Supervisor **Dr. Ruchi Verma (**Assistant Professor (SG), Department of CSE Jaypee University of Information Technology,Wakhnaghat). Deep Knowledge & keen interest of our supervisor in the field of our project **"Blind Mate Chess"** helped us to carry out this project. Her endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stages has made it possible for us to complete this project.

We would like to express our heartiest gratitude to **Dr. Ruchi Verma**(Assistant Professor (SG), Department of CSE), for her kind help to finish our project.

We would also generously welcome each one of those individuals who have helped us straightforwardly or in a roundabout way in making this project a win. In this unique situation, We might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated our undertaking.

Finally, We must acknowledge with due respect the constant support and patience of our parents.

Mr. Sheel Bhadra Joshi
(201175)

Mr. Mohit Verma
(201173)

ABSTRACT

Chess is one of the most popular board sports in the world currently. It has evolved to become a highly competitive and professional sport. In the 20th century International Federation of Chess(FIDE) was founded which organizes and governs chess tournaments and championships all around the world. Traditionally the game was played on a physical board with the presence of the players physically but as the Internet came, the world of chess took a major shift. Now players are able to compete against each other from anywhere in the world at any time without being physically present next to each other. According to an article **[1]** by chess.com as time went by many popular online chess platforms were founded such as **Chess.com** and **Lichess** and saw a growth of a huge number of registered players on their platform as the internet became more accessible to the common people. Around the mid 2000's independent tournaments started being organized on online chess platforms and later on FIDE also gave recognition to these platforms and started conducting several major championships on these platforms. It became very easy for players to participate in competitions from anywhere around the world. However a community of chess players consisting of people with visual impairments are still not able to find their place in the online chess world. They are not able to compete in online tournaments and championships as the platforms on which these championships take place do not provide any interface for blind people to access it. This does not provide them with equal opportunities to compete in the game.

To solve this problem the project aims to develop an application that would assist blind players to access these online chess platforms and play on them without any hassle. The application begins with recognizing the chessboard and pieces shown on the computer screen and then asking the player to prompt his/her move in the form of a voice prompt or hand gestures at his turn while simultaneously giving guidance and feedback to the player about the chessboard position, move played by the opponent and status of the game. The application gets the opponent moves and status of the game data using the Lichess API token generated by the player. After the move is prompted the application confirms the move with the player and calculates the board coordinates of the starting and ending position of the piece and finally plays that move by simulating mouse movements and clicks on the chessboard. The application also provides the player with various other features such as puzzles, game analysis, and a database of chess games.

Chapter-01 Introduction

1.1 Introduction

Chess is a beautiful game of strategy, intellect, calculation, patience and foresight. A beautiful aspect of chess is that even people with physical and visual disabilities can also enjoy the game to the fullest. Everyone has an equal and fair advantage in the game regardless of any disabilities. This is what makes it so inclusive and embracing in nature.

Chess players with visual impairment have been able to successfully shatter the perception that they are in any sense behind other players. These players are easily able to reconstruct the chessboard positions in their minds and are able to remember and also quite remarkably calculate the moves in their mind without being able to see a physical chessboard. In today's time many popular online chess platforms like **Lichess** and **Chess.com** are available with more than 100 million registered users on these platforms. Nowadays many of the important chess tournaments like International Bullet Chess Championship, Titled Tuesday etc. are being organized on these online chess platforms giving players the advantage to play in a comfortable environment of their choice without the need of being present physically. Online chess platforms are a very powerful tool to practice and improve in the game of chess as it allows you to play to a wide variety of players without being physically present.

However to this date blind players are not able to participate in online tournaments due to the lack of any interface to help or guide them. They still need to be physically present in front of their opponents to play a game of chess. This can be very time and money consuming. This has been a major drawback for blind individuals as they are unable to practice to their full potential and take part in various online chess tournaments and competitions. Our project aims to overcome this problem and develop an application that can be integrated with online chess platforms and help people with visual impairments to access these online chess platforms and participate in online tournaments and championships. It would also help them to practice and hone their skills as it would provide them with the opportunity to play with players from all around the world without being able to go anywhere. It would save a lot of time and money for people with visual impairments and allow them equal opportunity to participate in the sport.

1.2 Problem Statement

Chess has become a very popular game amongst people of every age group all over the world and most of the people enjoy this game with the help of online chess platforms such as lichess and chess.com. There are more than 110 million online users that are registered on these websites. The drawback of these platforms is that they do not provide any way or interface for people with visual impairments to access these applications and play on these platforms.

Our project aims to bridge the gap between online chess platforms and people with visual impairments and find a way to let the blind players have an engrossing, engaging experience of playing chess on online platforms.

Our problem statement is to develop an application that assists people with visual impairments to access online chess platforms, assisting them in playing online chess games without any hassle. The application should provide the blind users with real time update on chessboard positions, inform them about the opponents moves, current player with advantage in the game, ask the users the move they want to play in the game, reconfirm the moves being prompted by the user and accurately simulate those movements on the online chessboard in real time.

Our project aims on using image recognition techniques to accurately identify chessboard on the computer screen, using machine learning model to accurately predict the chess pieces and their positions on the board, use the API of the online Lichess platform to get the current moves being played by the opponent in the game, continuously assist the blind individual with voice feedback to inform the person about the game status, position updates, ask him for the moves he wants to play, get the moves of the user using the voice command or hand gestures, check if the moves being prompted are valid or not and finally use python library to simulate the mouse movement to play the desired move prompted by the user.

In conclusion our aim is to resolve the problem of playing online chess for people with visual impairments and thus promote inclusivity and competitiveness in online chess and contribute towards making the online chess community more diverse and accessible for all individuals regardless of any disabilities.

1.3 Objectives

The primary objective of this project is to develop an application that assists people with visual impairments to play chess on online chess platforms with an easy user interface and without any hassle. The detailed objectives of the project include:

1. **Accurately recognize chessboard on screen:** The first objective of this project is to accurately identify the chessboard that has been displayed on the screen using image recognition techniques to identify straight lines on the screen and make use of the fact that a chessboard has ten equally spaced horizontal and vertical lines present on it.

2. **Accurately identify the position on the chessboard:** The second objective of the project is to to accurately identify all the pieces that are present on the chessboard and recognize the current position on the chessboard.

3. **Real time chess game monitoring:** The third objective of the project is to monitor the ongoing game of the individual in real time so that current game status, positional advantage and move turn of the player can be known. The application should be able to tell the current position of the chessboard whenever the user asks for it.

4. **Voice feedback and guidance:** The fourth objective of the project is to continuously provide voice feedback and guidance to the individual about the status of the game, ongoing position of the game, whether the move prompted by the user is valid or not, whose turn is it to play in the game and what is the outcome of the game.

5. **Accurately recognize prompts from the user:** The fifth objective of the project is to to accurately recognize the chess moves or other commands prompted by the user in the form of hand gestures or voice commands and play the correct move or do the needful.

6. **Accurate chess move simulation:** The sixth objective of the project is to convert the chess notations prompted by the user into precise pixel coordinates on the screen. After identifying the correct pixel coordinates on the screen the application should simulate the mouse movements and clicks to execute the move on the board and move the chess piece from its original position to the target position.

7. **Provide game analysis and puzzles:** The final objective of the project is to provide game analysis to the player and help the player in evaluating a position by suggesting him best, worst and good moves and also to provide different categories of puzzles for the player along with their solutions.

1.4 Motivation and Significance

A research paper titled Problems of blind chess players published in the year 2015 **[2]** in the 6th IEEE International Conference on Cognitive Infocommunications discussed how blind players are left out from the world of online chess. There is no way for them to engage in online chess on popular online chess platforms such as Lichess and Chess.com. There has been little to no research conducted on the field of online chess for blind people. Most of the chess tournaments and championships that take place now are held on online chess platforms in which almost all professional chess players participate in but blind players are left out of these championships as the platforms on which these championships take place do not provide any tool for people with visual disabilities to assist them in playing online chess. The physical chess tournaments are held in different countries in different time zones, therefore it would require a lot of money and time to be present physically to play the tournament. In the covid time period physical chess tournaments were shifted on online chess platforms. In this time period blind players were left with no option to but abstain from participating in these tournaments. This is unfair to the sport of chess and to blind players. Our project aims to address this problem and make chess accessible for all.

The significance of this project is that it addresses the longstanding issue of exclusion of visually impaired individuals from the world of online chess. By making an application that breaks down the barrier between blind people and online chess players the project aims to empower and include people with visual impairments in the world of online chess. The project intends to provide equal opportunities for training, practicing and playing chess for people with visual impairments. The application will allow people with visual impairments to participate in online chess competitions and not let them be excluded from these tournaments which is the case currently. This project takes a significant step towards helping people with visual impairments to be independent to play chess on online platforms and provide an equal opportunity to compete with professional chess players in any online chess tournament.

1.5 Organization of Project Report

This project report is made in a way to provide a complete understanding of the Blind Mate Chess project, discussing its objectives, methodology, implementation, challenges, results, and future scope. The report is organized into six chapters, each dealing with a specific aspect of the project:

Chapter 1: Introduction

This chapter deals with introduction to the Blind Mate Chess project, providing a brief overview of its purpose, significance, and motivation. It highlights the objectives and problem statement of the project.

Chapter 2: Literature Survey

This chapter presents an overview of relevant literature and key gaps in the literature studied during the making of Blind Mate Chess application, covering existing research and developments in the field of chess boards and chess pieces detection.

Chapter 3: System Development

This chapter delves into the requirements and analysis phase of the project. It gives a detailed description of Project Design and Architecture, describes the Data preparation process and Implementation of the project along with the screenshots of the code snippets and finally Key challenges are addressed that were faced during the project.

Chapter 4: Testing

This chapter describes the testing methodology used in testing of the application, explaining the test cases and outcomes for various components of the code of the Blind Mate Chess application.

Chapter 5: Results and Evaluation

This chapter discusses the key findings of the project and their interpretation along with a snapshot of the results and evaluation metrics for detection of pieces, chessboards and hand gestures. Finally a comparison is performed with the existing solutions.

Chapter 6: Conclusion and Future Scope

This chapter concludes the project report by summarizing the key achievements, contributions and limitations of the Blind Mate Chess project. It also discusses potential future scope and enhancements for the application.

Chapter-02 Literature Survey

2.1 Overview of Relevant Literature

There has been very little research conducted in the field of making an application to assist blind individuals to play chess on online platforms. However there have been studies conducted in the field of physical chess board and chess pieces recognition so that moves played by the player over the board can be converted into a digital board for analysis and better visualization.

The techniques that have been mostly used in the research papers for accurate chessboard detection and chess pieces detection are Canny edge detection, RANSAC algorithm, CNN and Hough transform. The results have shown that these algorithms can achieve up to 99% accuracy and are very efficient in accurate detection of physical chess boards and pieces. These algorithms however have not been tested upon virtual chessboard and pieces detection. Upon extensive research we were still not able to find research papers on virtual chessboard and chess pieces detection. Virtual chess boards can have various different kinds of styles for chess boards as well as pieces as opposed to physical boards and pieces which generally have the same style therefore we cannot say for sure whether the algorithms would work accurately on virtual chess boards and pieces or not.

There has been development of voice activated physical chess boards that have been built for physically impaired people which use various electronic devices such as motors and microcontrollers to move the pieces and speech to text technologies for interaction with the chessboard but these devices do not provide the wide range of features and wide spectrum of opponents that are offered by online applications. Moreover these devices depend upon voice commands from the user to play the desired move but are not very effective in case of ambient noise.

The two major online chess platforms Chess.com and Lichess host the official chess tournaments and championships. Lichess platform provides an API token with the help of which we can fetch current game data of the player. The Lichess API documentation **[3]** mentions that the game data consists of the game status and moves played by the opponent.

The python-chess library **[4]** in python provides support for verifying whether the chess move is legal or not which can be useful to correct the blind player in case he/she tries to play an illegal move in the game.

Literature Review

S No.	Paper Title	Journal & Conference Year	Tools and Techniques	Results	Limitations
1.	Determining chess game state from image **[5]**	Journal of imaging (2021)	Canny edge detection and CNN	Accuracy: 92%	Model may misidentify chess pieces
2.	Chessboard and chess piece recognition with the support of neural network **[6]**	Foundations of computing and decision science (2020)	Hough lines transform and CNN	Accuracy: 95%	Slower than alternative approaches
3.	Augmented reality chess analyzer **[7]**	Journal of emerging investigators (2020)	Canny edge detection, hough lines transform and CNN	Accuracy: 93%	Algorithm is only designed to work with physical chessboards and pieces
4.	Robust computer vision chess analysis and interaction with humanoid robot **[8]**	IEEE international conference on control, automation and robotics (2019)	Canny edge detection and hough lines transform	Accuracy: 95%	Algorithm may be less accurate if the chessboard is not placed in a well lit area
5.	Chess position identification using pieces classification based on deep neural network fine-tuning **[9]**	21st Symposium on virtual and augmented reality (2019)	Canny edge detection, hough lines transform and VGG16	Accuracy: 96%	Model may not be able to predict chess pieces accurately at different camera angles

Literature Review

S No.	Paper Title	Journal & Conference Year	Tools and Techniques	Results	Limitations
6.	Chessvision: chess board and piece recognition **[10]**	Stanford journal of science, technology and society (2019)	Canny edge detection and SVM	Accuracy: 99%	Likely to make mistakes if the images are blurry
7.	CVChess: computer vision chess analytics **[11]**	Stanford journal of science, technology and society (2018)	Harris corner detection and RANSAC algorithm	Accuracy: 99%	Algorithm is computationaly expensive
8.	Chess piece recognition using oriented chamfer matching with comparison to CNN **[12]**	IEEE winter conference on applications of computer vision (2018)	Canny edge detection, hough lines transform and oriented chamfer matching	Accuracy: 95%	Algorithm is not able to detect occluded chess pieces accurately
9.	Chess recognition from a single depth image **[13]**	IEEE international conference on multimedia and expo (2017)	CNN using data from kinect v2 sensor	Accuracy: 90%	Proposed algorithm is sensitive to different camera angles and lighting conditions
10.	Geometry based populated chessboard recognition **[14]**	The 10th international conference on machine vision and machine learning (2017)	Canny edge detection and hough lines transform	Accuracy: 99%	Model may not be able to predict chess pieces accurately at different camera angles

Literature Review Table

S No.	Paper Title	Journal & Conference Year	Tools and Techniques	Results	Limitations
11.	A computer vision system for chess game tracking **[15]**	IEEE winter conference on applications of computer vision (2016)	Canny edge detection and CNN	Accuracy: 99%	Model is sensitive to lighting conditions and camera angles
12.	Visual chess recognition **[16]**	Stanford journal of science, technology and society (2015)	Hough transform and fourier descriptor	Accuracy: 90%	Dataset used to train and evaluate the algorithm is relatively small
13.	Gambit: a robust chess playing robotic arm **[17]**	IEEE international conference on control, automation and robotics (2011)	RANSAC algorithm and CNN	Accuracy: 93%	Model is only able to detect standard chess pieces
14.	A simple autonomous robotic manipulator for playing chess against any opponent in real time **[18]**	International conference on computational vision and robotics (2011)	Shi-Tomasi corner detection and canny edge detection	Accuracy: 95%	Algorithm is tested on a limited dataset
15.	Automatic chessboard detection for intrinsic and extrinsic camera parameter calibration **[19]**	Sensors (2010)	Harris corner detection and hough transform	Accuracy: 93%	Algorithm is only designed to work with physical chessboards and pieces

2.2 Key Gaps in Literature

1. **No integration with popular online chess platforms:** The very few voice based chess applications for blind people that exist mostly have only the feature of playing chess with a computer. Most of the tournaments that take place for chess are organized on two famous online chess platforms Lichess and Chess.com. These two platforms have the highest number of registered chess users and all the professional chess players use these platforms to practice and train their skills. Moreover these platforms allow players to play with people all over the world. Standalone voice based chess applications that only allow players to play with a computer do not provide the benefits of online chess.

2. **Insufficient voice guidance and feedback:** The voice based chess applications that exist do not provide sufficient guidance and commentary for the person with visual impairments. To play the game in an efficient manner and to remember complex chess positions correctly the individual needs a time to time guidance and update about the position when asked for and also the current situation of the game. The applications that exist do not provide the current scenario of the chessboard positions every time the user asks for it. They only update the user about the current move played in the game. Most of the time it can happen with the individual that he or she forgets the current position of pieces in the game. In such a case he needs to be again guided by the voice guidance about the current position of pieces on the chessboard.

3. **Lack of research on recognizing a virtual chessboard:** Most of the research that has been conducted regarding the accurate recognition of chessboards has been done and tested with physical chess boards. The pieces and board orientations can be very different in the case of a virtual chessboard as opposed to a physical board. A lot more research is needed for recognizing a virtual chessboard which will eventually help in making an application to assist people with visual impairments to play the game of chess on online platforms.

4. **Issues in voice command recognition:** Voice assisted chess applications rely on accurate voice recognition features to interpret the prompt given by the user. In online chess the player is required to prompt the moves he wants to play in the game. In such a case the voice recognition should be accurate because once a move is played it can’t be taken back. The effectiveness of voice recognition can be hampered if there is ambient noise. The existing solutions assume an ideal acoustic environment and neglect the potential real life scenarios.

Chapter-03 System Development

3.1 Requirements and Analysis

Language Used: Python 3.11.2

Technical Requirements:

- A computer with at least 4GB of RAM and a multicore processor
- Internet Connection

Software:

- Python 3.5 or higher
- Visual Studio Code or any other code editor

Libraries:

- requests (making HTTP requests)
- json (handling JSON data)
- PIL (image processing)
- numpy (numerical computing)
- math (mathematical operations and functions)
- cv2 (computer vision library)
- pyautogui (automating mouse movements)
- chess (evaluating chess positions)
- whisper (speech text transcription)
- wave (handling audio data)
- os (interacting with operating system)
- warnings (handling warnings and exceptions)
- speech_recognition (speech recognition library)
- re (regular expression pattern matching)
- pyttsx3 (text to speech library)
- YOLO (YOLO object detection model)
- mediapipe (detecting hand landmarks)

Additional Requirements

- stockfish chess engine (analyzing the games)

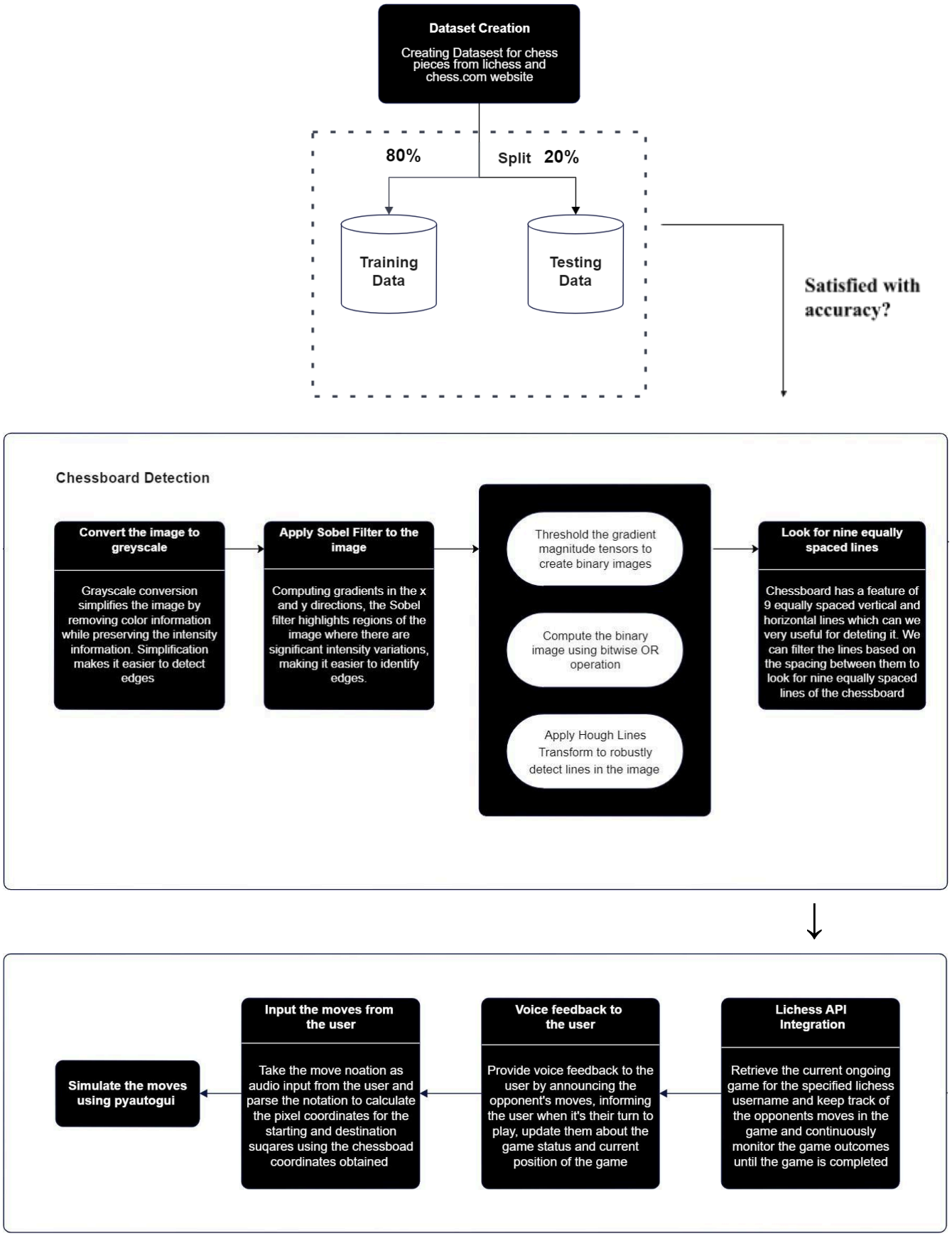
Functional Requirements

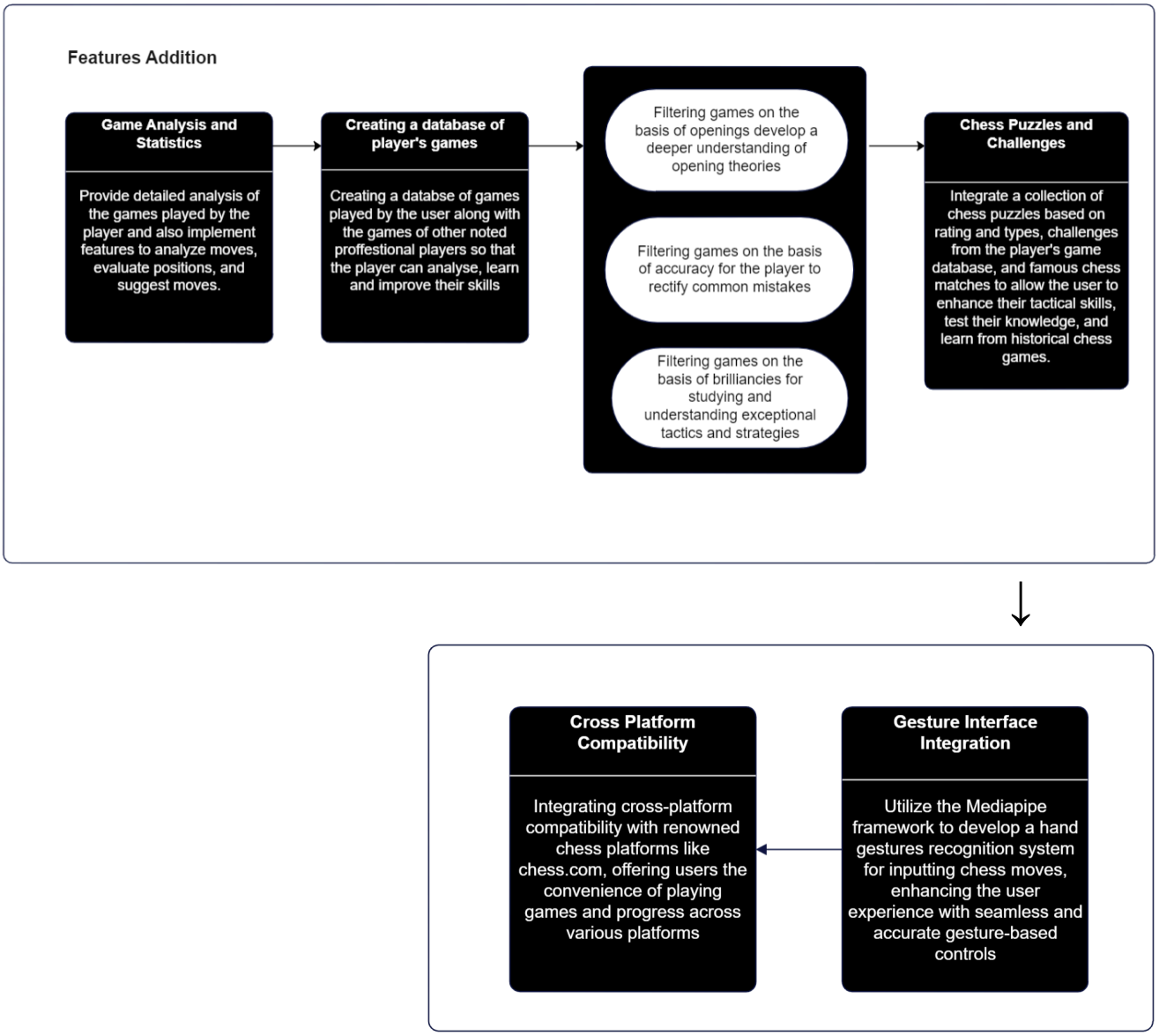
- The application should be able to accurately identify the chessboard and distinguish between different types of pieces from the screenshot of the screen.
- The application should be able to accurately transcribe move notations and prompts from the player.
- The application should be able to accurately identify the gestures shown by the player.
- The application should be able to assist the player by providing the game status, current chess board position and inform the player about his turn.
- The application should be able to calculate coordinates of starting and ending squares of the notation and simulate the mouse movements accordingly.
- The application should be able to notify the player when he/she tries to play an illegal move.
- The application should be able to provide puzzles and analysis of games to the player.

Non Functional Requirements

- The application should be able to process image and speech input in real-time with minimum latency to provide a real time gameplay experience.
- The application should be able to detect the chessboard and transcribe the prompt with maximum accuracy to ensure a desired gameplay experience.
- The application should be stable and reliable, minimizing crashes or unexpected behavior.
- The application should be compatible with different types of devices, including smartphones, tablets, and computers.
- The application should be able to adapt to different accents, speech patterns, and languages, providing accurate speech recognition.
- The application should be able to adapt to different camera angles, lighting conditions, and hand orientation, providing accurate gesture recognition.

3.2 Project Design and Architecture




Features Addition
Game Analysis and Statistics
Provide detailed analysis of the games played by the player and also implement features to analyze moves, evaluate positions, and suggest moves.
Creating a database of player's games
Creating a databse of games played by the user along with the games of other noted proffestional players so that the player can analyse, learn and improve their skills
Filtering games on the basis of openings develop a deeper understanding of opening theories
Filtering games on the basis of accuracy for the player to rectify common mistakes
Filtering games on the basis of brilliancies for studying and understanding exceptional tactics and strategies
Chess Puzzles and Challenges
Integrate a collection of chess puzzles based on rating and types, challenges from the player's game database, and famous chess matches to allow the user to enhance their tactical skills, test their knowledge, and learn from historical chess games.
Cross Platform Compatibility
Integrating cross-platform compatibility with renowned chess platforms like chess.com, offering users the convenience of playing games and progress across various platforms
Gesture Interface Integration
Utilize the Mediapipe framework to develop a hand gestures recognition system for inputting chess moves, enhancing the user experience with seamless and accurate gesture-based controls

1. Firstly we take a screenshot of the current window screen on which the blind person is playing his current game of chess. Then we convert the screenshot(PIL image) to a numpy array to make it compatible with the OpenCV functions. After that we convert the colored image of the screenshot to a grayscale image which removes the color information of the image while preserving its intensity information. This helps in removing extra information from the image and making it easier to detect edges in the image which is an important step towards line detection in the image.

2. Next we apply a sobel filter on the grayscale image in both horizontal(x) and vertical(y) directions. Edges in an image represent those areas where there is a rapid change in the intensity or color. The Sobel filter aids us in capturing the intensity changes in the image in both horizontal(x) and vertical(y) directions. The Sobel filter highlights the regions of the image where there are significant intensity variations by computing the gradients in the x and y directions which makes it easier to identify the edges. After this we calculate the absolute values of the gradients in both horizontal and vertical directions which results in two gradient magnitude tensors.

3. After computing the gradient magnitude we obtain an image where each pixel represents the strength of the edge at that particular point on the image. To identify and isolate significant edges in the image we need to distinguish between strong and weak edges. Thresholding helps us in creating a binary image that highlights the regions which have edges of certain strengths and suppress the regions with weaker edges. The output image of this step is an image where each pixel in the image is assigned with one of the two values, either 0(black) or 255(white). The binary images we obtain are combined using Bitwise OR operation and we get a single binary image called “edge map” which has edge information from both x and y gradient directions.

4. Finally we apply Hough Transform on the edge map which is a mathematical technique to detect geometrical shapes particularly lines in an image. The Hough Transform represents lines in the hough space which is parameterized by the polar coordinates ρ (rho) which is the distance from the origin to the closest point on the line and θ (theta): The angle between the x-axis and the line perpendicular to the line being represented. We use the **HoughLinesP** function to implement the Hough Transform. The function takes parameters such as minLineLength, maxLineGap which is the maximum allowed gap between line segments to treat them as a single line, threshold which is the number of votes a line should get in the accumulator array to be considered a valid line.

5. After the lines are detected in the image we look for lines which are at equal distance from each other. In a chessboard there are nine equally spaced lines present in both horizontal and vertical direction. If we get those nine equally spaced lines in the image we are correctly able to identify the chessboard and its position on the computer screen.

6. Next we take an image of each square of the chessboard and apply a YOLO **[20]** object detection algorithm to identify the chess piece present on that square. YOLO which stands for You Only Look Once is an object detection algorithm that makes predictions with bounding boxes and class probabilities for an image in a single pass. The YOLO model gives a vector as its output for each object detected in an image and the vector contains bounding box coordinates (x,y,width,height) for each detected object in the image, object class probabilities and an objectness score.

7. Then we integrate the application with the popular Lichess platform using the API key of the player and his username which helps us to fetch the data of the current ongoing game of the player and keep a track of the moves played by the opponent in the game. It also helps us to get an update of the current status of the game, such as the winner of the game or if there is any resignation in the game.

8. Afterwards we include the Whisper AI model **[21]** which is an open source general purpose speech recognition model. It has been said to have been trained on 6,80,000 hours of labeled audio recordings available on the internet which include audios in different languages. A big and varied set of data makes the model better at understanding different accents and background noises. A study comparing Whisper and six other speech recognition models concluded that Whisper was able to outperform the most widely used open source model NVIDIA STT on all the tested datasets. We use it to accurately identify the move notations and prompts from the player.

9. We then use pyttsx3 library in Python to give voice feedback to the user and ask him/her to prompt the move the user wants to play. Once the user prompts the move it is transcribed using Whisper AI and the application asks the user to confirm the move he/she wants to play. Once the move is successfully recognized we convert it into precise pixel coordinates on the screen which are determined by the current configuration of the chessboard. Subsequently the application simulates the mouse movements and clicks to execute the move and relocate the piece from its starting position to the desired destination.

10. We then implement a hand gesture recognition system for taking prompts from the user. We make the use of mediapipe **[22]** which is an open source framework that was developed by Google to build machine learning pipelines. The Hand Landmark model is trained on thirty thousand real world images and it also works effectively on occluded or partially visible hands. We use it to track hands and locate hand landmarks. Each hand landmark represents a specific point on the hand. We then iterate over specific landmarks and check their positions with respect to other landmarks. To check if a finger is open or closed we will compare the Y-coordinate of the finger_tip landmark and finger_pip landmark. Whenever the finger will be upwards the y coordinate of the finger_tip landmark will have a lower value than the finger_pip landmark. We then create custom gestures to represent columns and rows in a chessboard along with other requests.

11. We then fetch all the games of a user from both Lichess and Chess.com using API and create a database out of it in which the games are filtered on the basis of various factors such as accuracy, openings, brilliancies, time format etc. This helps in providing the user with useful statistics and information in a clear manner. It can help the user to understand his/her gameplay in a better manner. The user can also look up in the database for any game he/she wants to analyze and can analyze it at any time.

12. Then we implement a feature for daily puzzles and fetch the daily puzzle from Lichess and also create a database containing puzzles from positions that were encountered in famous games. The puzzles also contain a wide variety of scenarios such as “mate in 1”, “zugzwang, fork” etc.

13. Finally we include engine analysis for the user to provide the user with valuable insights of their game. With the help of a very powerful chess engine called “stockfish” the users can evaluate any chess position and can choose to try and play moves in that particular position and understand the move advantage or disadvantage in that position. The stockfish engine provides a score of the chessboard position in the form of centipawn which is equivalent to one hundredth of a pawn and in the case of mate it represents it with a “#” followed by a number which denotes the number of moves in which the checkmate is due. The engine also gives the best n moves in any position for a player. The player can prompt his own move in a position and analyze the evaluation of the position before and after his move. This helps the user to compare various moves and identify the best, worst and good moves in a position.

3.3 Data Preparation

Data Collection:

- Collected images of chess boards and chess pieces from various online chess platforms such as Chess.con, Lichess, Chess24.com etc.

- Ensured images cover a diverse range of chess piece and chess board styles with different colors and backgrounds including traditional wooden chess boards, modern plastic chess boards and chess boards with unique designs or colors.

- Captured images of chess pieces in different orientations, including upright, flipped, and partially occluded.

- Took around 250-300 images of each chess piece and chess boards with different styles and backgrounds.

- Collected images of various hand gestures for testing the hand gesture recognition algorithm.

Data Labeling:

- Manually labeled each image to identify different types of chess pieces in each square.

- Used appropriate annotation tools to create bounding boxes chess pieces.

Data Splitting:

- Divided the dataset into two training and testing subsets. 80% of the data was used for training and the rest 20% of the data was used for testing.

3.4 Implementation

```
import requests
import json
import sys
from PIL import Image, ImageOps
import numpy as np
from keras.models import load_model
import math
import cv2
import pyautogui
import chess

import whisper
import wave
import os
import warnings

import speech_recognition as sr
import re
import pyttsx3



warnings.filterwarnings("ignore", message="FP16 is not supported on CPU")
api_token = "lip_yzGgMK9RQgJTo2aMUlHH"
username = "Happy02001"
```

Snippet 3.4.1

The above code is used for importing necessary libraries for the application to run such as requests for making HTTP requests, PIL for image processing, pyautogui library **[23]** for mouse movements simulation etc and it also contains the api token and username of the blind player for Lichess platform which will be needed to fetch game data from the platform.

```
screenshot = pyautogui.screenshot()

img = np.array(screenshot)

img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

Dx = cv2.Sobel(img, cv2.CV_64F, 1, 0)
Dy = cv2.Sobel(img, cv2.CV_64F, 0, 1)

mag_x = np.abs(Dx)
mag_y = np.abs(Dy)

thresh_x = cv2.threshold(mag_x, 100, 255, cv2.THRESH_BINARY)[1]
thresh_y = cv2.threshold(mag_y, 100, 255, cv2.THRESH_BINARY)[1]

edge_map = cv2.bitwise_or(thresh_x, thresh_y)

edge_map = cv2.convertScaleAbs(edge_map)
lines = cv2.HoughLinesP(edge_map, 1, np.pi/180, 200, minLineLength=200, maxLineGap=4)
```

Snippet 3.4.2

```
filtered_lines = []
bottommost_coords = []
bottommost_coords_y = []
topmost_coords = []
topmost_coords_y = []
x_diffs = []

min_distance = 2

for line in lines:
    x1, y1, x2, y2 = line[0]
    angle = np.arctan2(y2 - y1, x2 - x1) * 180 / np.pi
    if 80 < angle < 100 or -100 < angle < -80:
        filtered_lines.append(line)
        bottommost_x = max(x1, x2)
        bottommost_y = max(y1, y2)
        topmost_x = min(x1, x2)
        topmost_y = min(y1, y2)
        bottommost_coords.append(bottommost_x)
        bottommost_coords_y.append(bottommost_y)
        topmost_coords.append(topmost_x)
        topmost_coords_y.append(topmost_y)


bottommost_coords.sort()
bottommost_coords_y.sort()
topmost_coords.sort()
topmost_coords_y.sort()
```

Snippet 3.4.3

In the above code we filter the vertical lines and the coordinates are sorted in ascending order to calculate the difference between consecutive coordinates of the detected lines.

```
prev_diff = None
similar_y_coords_bot = []
similar_y_coords_top = []
count = 0

for i in range(len(bottommost_coords) - 1):
    if len(similar_y_coords_bot) == 8:
        break
    x_diff = bottommost_coords[i + 1] - bottommost_coords[i]
    x_diffs.append(x_diff)

    if x_diff is not None and prev_diff is not None and abs(x_diff - prev_diff) <= 1:
        similar_y_coords_bot.append(bottommost_coords_y[i])
        similar_y_coords_top.append(topmost_coords_y[i])
        count += 1
        if count == 1:
            similar_y_coords_bot.append(bottommost_coords_y[i])
            similar_y_coords_top.append(topmost_coords_y[i])

    prev_diff = x_diff

for line in filtered_lines:
    x1, y1, x2, y2 = line[0]
    cv2.line(img, (x1, y1), (x2, y2), (0, 255, 0), 2)
    cv2.circle(img, (x1, y1), 5, (0, 0, 255), -1)
    cv2.circle(img, (x2, y2), 5, (0, 0, 255), -1)

BOARD_SIZE = bottommost_coords[len(bottommost_coords)-1]-bottommost_coords[0]
CELL_SIZE = int(BOARD_SIZE / 8)
BOARD_LEFT_COORD = bottommost_coords[0]
BOARD_TOP_COORD=similar_y_coords_top[0]

print(f"Board size: {BOARD_SIZE}, Cell size: {CELL_SIZE}, Top coordinate: {BOARD_TOP_COORD}, Left coordinate: {BOARD_LEFT_COORD}")
```

Snippet 3.4.4

In the above code, the difference between consecutive x coordinates and consecutive y coordinates are calculated for all the lines that were detected in the previous step. The lines that have the same difference between consecutive x coordinates and consecutive y coordinates are appended in the list. The goal is to look for nine such lines that are almost equidistant to one another since it is the main feature that helps in identifying a chessboard. The chessboard is made up of nine equally spaced horizontal and vertical lines containing sixty four squares.

```
model = YOLO("yolov5nu.pt")
class_names = [
    "black_pawn",
    "black_king",
    "black_bishop",
    "black_queen",
    "black_rook",
    "black_knight",
    "white_pawn",
    "white_rook",
    "white_bishop",
    "white_knight",
    "white_king",
    "white_queen",
    "chessboard",
    "empty_squares"
]
```

Snippet 3.4.5

In the above code a YOLOv5 model instance is created and the path to the model weights are passed as an argument. A list containing piece names is defined that the YOLO model can detect.

```
def create_board():
    board = np.zeros((8, 8), dtype=str)
    for row in range(8):
        for col in range(8):
            cell_top_coord = BOARD_TOP_COORD + row * CELL_SIZE
            cell_left_coord = BOARD_LEFT_COORD + col * CELL_SIZE
            screenshot = pyautogui.screenshot()
            cell_image = screenshot.crop(
                (cell_left_coord, cell_top_coord, cell_left_coord + CELL_SIZE, cell_top_coord + CELL_SIZE)
            )
            cell_image_grayscale = cell_image.convert("L")
            cell_image_resized = cell_image_grayscale.resize((224, 224), resample=Image.BILINEAR)
            cell_image_rgb = cell_image_resized.convert("RGB")
            cell_image_array = np.asarray(cell_image_rgb)
            data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)
            data[0] = cell_image_array / 255.0
            prediction = model.predict(data)
            index = np.argmax(prediction)
            class_name = class_names[index]
            percentage = round(prediction[0][index] * 100, 2)
            piece = class_name.split("_")[1]
            color = class_name.split("_")[0]
            piece_code = 0
            if piece == "pawn":
                piece_code = 'P' if color == "white" else 'p'
            elif piece == "rook":
                piece_code = 'R' if color == "white" else 'r'
            elif piece == "knight":
                piece_code = 'N' if color == "white" else 'n'
            elif piece == "bishop":
                piece_code = 'B' if color == "white" else 'b'
            elif piece == "queen":
                piece_code = 'Q' if color == "white" else 'q'
            elif piece == "king":
                piece_code = 'K' if color == "white" else 'k'
            elif class_name == "empty_squares":
                piece_code = '.'
            board[row][col] = piece_code
    print(board)
    return board
```

Snippet 3.4.6

The above code firstly we create a empty 2d numpy array to represent a chessboard with 8*8 dimensions. Each element in the array will contain the piece present in the corresponding square of the virtual chessboard. Next we iterate over each square of the chessboard, calculate the coordinates of the square and crop the square image for identification of the chess piece lying on the square. The cropped image is converted into grayscale and a YOLO model is applied to predict the piece on the square. Once the piece is predicted it is mapped to chess notation code. Eg. A white pawn is represented by ‘P’ and a black pawn is represented by ‘p’. The determined piece is then assigned to the corresponding position in the chessboard array. The generated chessboard with the piece code is printed on to the console.

```python
def get_current_ongoing_game(username):
    try:
        headers = {
            "Accept": "application/json"
        }
        response = requests.get(f"https://lichess.org/api/user/{username}/current-game", headers=headers)
        if response.status_code == 200:
            game_data = response.json()
            return game_data
        else:
            print(f"Failed to retrieve the current ongoing game for {username}.")
            return None
    except requests.RequestException as e:
        print(f"An error occurred while retrieving the current ongoing game: {str(e)}")
        return None
```

Snippet 3.4.7

The above code defines a function named get_current_ongoing_game() that takes the username of the player as an input and attempts to retrieve the current ongoing game for that user from the Lichess API. The function uses the 'requests.get' method to make a GET request to the lichess API endpoint for retrieving the current game information. The function then checks the HTTP status code. If the request is successful (status code 200), it parses the JSON response using response.json() to obtain game data and if the status code is different from 200, an error message is printed on to the console. If it is successful then the function returns the game data. The above code also handles any kind of network errors or exceptions using a try-except block, which is used for catching exceptions of type requests. RequestException and printing the error message before returning None. The game data received is of the following form.

{'id': 'VLrDU8Zp', 'rated': False, 'variant': 'standard', 'speed': 'correspondence', 'perf': 'correspondence', 'createdAt': 1715487615065, 'lastMoveAt': 1715487649836, 'status': 'started', 'source': 'ai', 'players': {'white': {'user': {'name': 'Happy02001', 'id': 'happy02001'}, 'rating': 1500, 'provisional': True}, 'black': {'aiLevel': 1}}, 'opening': {'eco': 'C20', 'name': "King's Pawn Game: Leonardis Variation", 'ply': 3}, 'moves': 'd3 e5 e4 Nf6 f3 Bc5 Be2 Nh5 Bd2 Nc6 Nc3 d6 Nh3 O-O Na4 Bxh3 Nxc5', 'division': {'middle': 18}}

```python
game_id=""

if current_ongoing_game:
    game_id = current_ongoing_game['id']
```

Snippet 3.4.8

In the above code an empty string variable game_id is initialized. This variable will be used to store the unique game id of the current ongoing game. The code then checks if the current_ongoing_game variable is not None. If the current_ongoing_game variable is not None it means that the API call was successful and the game data was retrieved. Now the code extracts the current ongoing game's id which will be used in the further process of extracting the relevant game data.

```
def stream_game_moves(api_token, game_id):
    url = f"https://lichess.org/api/board/game/stream/{game_id}"
    headers = {
        "Authorization": f"Bearer {api_token}"
    }

    response = requests.get(url, headers=headers, stream=True)



    if response.status_code == 200:
        last_move = None
        for line in response.iter_lines():
            if line:
                game_data = line.decode("utf-8")
                game_json = json.loads(game_data)

                def check():
                    if game_json.get('status') == 'mate':
                        winner = game_json.get('winner', 'none')
                        if winner == 'white':
                            print("White player won the game.")
                            engine.say("White player won the game.")
                            engine.runAndWait()
                            sys.exit()

                        elif winner == 'black':
                            print("Black player won the game.")
                            engine.say("Black player won the game.")
                            engine.runAndWait()
                            sys.exit()
```

Snippet 3.4.9

```
                elif game_json.get('status') == 'resign':
                    winner = game_json.get('winner')
                    if winner == 'black':
                        print("White player resigned.")
                        engine.say("White player resigned.")
                        engine.runAndWait()
                        sys.exit()


                    elif winner == 'white':
                        print("Black player resigned.")
                        engine.say("Black player resigned.")
                        engine.runAndWait()
                        sys.exit()



                elif game_json.get('status') == 'draw':
                    print("Game ended in a draw")
                    engine.say("Game ended in a draw")
                    engine.runAndWait()
                    sys.exit()
```

Snippet 3.4.10

```
            if 'white' in game_json and 'black' in game_json:
                if game_json['white'].get('name') == username:
                    color = 'white'
                elif game_json['black'].get('name') == username:
                    color = 'black'
                else:
                    color = 'unknown'

            if 'state' in game_json and 'moves' in game_json['state']:
                moves = game_json['state']['moves']
                moves_count = len(moves.split())
                if moves_count > 0:
                    last_move = moves.split()[-1]
                else:
                    last_move=None
            elif 'moves' in game_json:
                moves = game_json['moves']
                moves_count = len(moves.split())
                if moves_count > 0:
                    last_move = moves.split()[-1]
                else:
                    last_move=None
            else:
                continue
```

Snippet 3.4.11

```
            if color=='white' and moves_count%2==0:
                check()
                create_board()
                print("Your turn")
                print(f"Opponent played: {last_move}")
                engine.say(f"Opponent played {last_move}, Your Turn")
                engine.runAndWait()

                move(last_move, color)


            elif color=='black' and moves_count%2==1:
                check()
                create_board()
                print("Your turn")
                print(f"Opponent played: {last_move}")
                engine.say(f"Opponent played {last_move}, Your Turn")
                engine.runAndWait()
                move(last_move, color)




        check()



    else:
        print(f"Error: {response.status_code} - {response.text}")

stream_game_moves(api_token, game_id)
```

Snippet 3.4.12

In the above code a function is defined named stream_game_moves() which takes an API token for authentication and a game ID which is the unique identifier of any chess game happening on Lichess platform as input and establishes a live stream connection with the help of Lichess API to receive real-time updates for the current ongoing game of the blind person. The function constructs an API url with the help of f strings using the game id obtained in the previous step. It then sets the required HTTP header which is the authorization header with the Lichess API token of the player.

The function then sends a HTTP GET request to the Lichess API endpoint using the requests.get() method for streaming the moves of the opponent with the help of API token and establishes a streaming connection to get real time updates. Once the data is received the function iterates through each line of the response using response.iter_lines().

Next there is a nested function check() defined which is for checking the current game status. The function checks if the current game status is mate, resign or draw and then identifies the winner of the game. Then the engine.say() method is used to declare to the blind person the winner of the game or a resignation or a draw message in case if the game is drawn or any of the opponents resigned. After the game has ended the program exists.

The function then determines the color of the pieces for the blind player by comparing it with the username of the player with the JSON data. It then counts the number of moves that have been played in the game to determine whose turn it will be to play the next move. If the moves count is even then it is white players turn to play the move or else if the move count is odd it is black players turn to play the move. The function determines the player's turn and then announces to the blind player about the opponent's last move and asks the opponent to play his/her move.

After announcing the opponents moves and asking the blind player to play, the move() function is invoked for further proceedings. The function meanwhile continuously checks the game status changes for any kind of draw, resignation or checkmate and if there is any error occurred during the streaming process it prints an error message. The streamed moves data is returned in the following manner.

```
{'id': 'VLrDU8Zp', 'variant': {'key': 'standard', 'name': 'Standard', 'short': 'Std'}, 'speed': 'correspondence', 'perf': {'name': 'Correspondence'}, 'rated': False, 'createdAt': 1715487615065, 'white': {'id': 'happy02001', 'name': 'Happy02001', 'title': None, 'rating': 1500, 'provisional': True}, 'black': {'aiLevel': 1}, 'initialFen': 'startpos', 'type': 'gameFull', 'state': {'type': 'gameState', 'moves': 'd2d3 e7e5 e2e4 g8f6 f2f3 f8c5 f1e2 f6h5 c1d2 b8c6 b1c3 d7d6 g1h3 e8g8 c3a4 c8h3 a4c5 d8e7 g2g3 h3g2', 'wtime': 2147483647, 'btime': 2147483647, 'winc': 0, 'binc': 0, 'status': 'started'}}
{'type': 'gameState', 'moves': 'd2d3 e7e5 e2e4 g8f6 f2f3 f8c5 f1e2 f6h5 c1d2 b8c6 b1c3 d7d6 g1h3 e8g8 c3a4 c8h3 a4c5 d8e7 g2g3 h3g2 e1f2', 'wtime': 2147483647, 'btime': 2147483647, 'winc': 0, 'binc': 0, 'status': 'started'}
```

```
def move(last_move, color):

    start_position = go(last_move)
    end_position = go(last_move)

    if color == "white":
        start_col = ord(start_position[0]) - ord('a')
        start_row = 8 - int(start_position[1])
        end_col = ord(end_position[0]) - ord('a')
        end_row = 8 - int(end_position[1])
    else:
        start_col = ord('h') - ord(start_position[0])
        start_row = int(start_position[1]) - 1
        end_col = ord('h') - ord(end_position[0])
        end_row = int(end_position[1]) - 1

    start_x = BOARD_LEFT_COORD + start_col * CELL_SIZE + CELL_SIZE // 2
    start_y = BOARD_TOP_COORD + start_row * CELL_SIZE + CELL_SIZE // 2
    end_x = BOARD_LEFT_COORD + end_col * CELL_SIZE + CELL_SIZE // 2
    end_y = BOARD_TOP_COORD + end_row * CELL_SIZE + CELL_SIZE // 2

    pyautogui.moveTo(start_x, start_y)
    pyautogui.mouseDown()
    pyautogui.moveTo(end_x, end_y)
    pyautogui.mouseUp()
```

Snippet 3.4.13

The move() function is an important component of the application, responsible for the execution of moves on the virtual chessboard. In the above code a function move is defined which takes two parameters: last move of the opponent and color of the pieces of the blind player. Then the function calls another go() function which will be used to ask the user for the starting position of the square from which he/she wants to move the piece from and also the ending position of the square on which the piece is supposed to go. After the user gives the notation the function based on the color of the blind player it converts the algebraic notation of the move prompted by the player into row and column indices. For white it uses the standard chessboard orientation and for black it flips the orientation to match the perspective. It then calculates the pixel coordinates for the start and end positions on the chessboard based on predefined constants such as **BOARD_LEFT_COORD**, **BOARD_TOP_COORD**, and **CELL_SIZE**. Finally the function makes use of the pyautogui library to simulate the mouse movements based on the calculated pixel coordinates and move the piece from the starting square of the chessboard to the desired square of the chessboard.

```
def go(last_move):
    notation = None
    while notation is None:
        notation = recognize_notation(last_move)

    return notation
```

Snippet 3.4.14

```
def recognize_notation(last_move):

    model = whisper.load_model("tiny.en")

    while True:
        with sr.Microphone() as source:
            print("Please say a valid chess notation")
            engine.say("Please say a valid chess notation")
            engine.runAndWait()

            r.adjust_for_ambient_noise(source)

            try:
                audio = r.listen(source, timeout=600, phrase_time_limit=3)
                audio_filename = "captured_audio.wav"
                with open(audio_filename, "wb") as f:
                    f.write(audio.get_wav_data())

                result = model.transcribe(audio_filename)

                text = result["text"]

                square_pattern = r'[a-hA-H][1-8]'
                chess_squares = re.findall(square_pattern, text)

                notation=""
```

Snippet 3.4.15

In the above code firstly the function go is defined in which variable notation is set to None and until the notation does not get a value the while loops keep on running and invokes the function recognize_notation. In the recognize_notation() function firstly the Whisper model is loaded which will be used to transcribe spoken english speech to text. Then the function enters a loop where it announces the player to say the chess notation and continuously listens for prompts from the player. When the audio is detected it saves the audio file temporarily with the extension .wav and then the whisper model transcribes the audio into text. Afterwards with the help of regular expressions the transcribed text is searched for a valid chess notation and if a valid chess notation is found then it is stored in the notation variable.

```
            if len(chess_squares):
                notation = chess_squares[0]

            if os.path.exists(audio_filename):
                os.remove(audio_filename)

            if notation:
                print(f"Recognized notation: {notation.upper()}")
                engine.say(f"You said {notation}, is that correct?")
                engine.runAndWait()
                confirm = recognize_confirmation(r, engine, count)

                if confirm:
                    return notation.lower()
                else:
                    engine.say(f"Last Move is {last_move}")
                    engine.runAndWait()
            else:
                print(f"Invalid notation: {text}")
                engine.say("Sorry, I did not recognize a valid notation")
                engine.runAndWait()

        except sr.WaitTimeoutError:
            print("Timeout occurred while waiting for phrase. Please speak again.")
            engine.say("Timeout occurred while waiting for phrase. Please speak again.")
            engine.runAndWait()

        except sr.UnknownValueError:
            print("Could not understand audio")
            engine.say("Sorry, I did not understand what you said")
            engine.runAndWait()

        except sr.RequestError as e:
            print(f"Could not request results from Google Speech Recognition service; {e}")
            engine.say("Sorry, I could not understand what you said")
            engine.runAndWait()
```

Snippet 3.4.16

After the notation is recognized the above code announces the player of the notation that has been recognized by the model. It then calls the recognize_confirmation function to get confirmation from the player for the recognized chess notation. If the player does not confirm the notation then the application announces to the user for incorrect recognition and again repeats the process of listening for a valid chess notation from the user. The function also includes error handling in case there is a timeout, or a valid audio is not recognized. This process involves handling various exceptions such as timeouts **(sr.WaitTimeoutError)**, unrecognized audio inputs **(sr.UnknownValueError),** and errors in requesting results from the Google Speech Recognition service **(sr.RequestError**). In all such cases the application announces to the player the following errors and again asks the player for a valid chess notation. The function asks for the destination square chess notation, after receiving the notation it calls the move_legal function to check if the player has prompted an illegal chess notation or not. If the player has prompted for a legal chess notation it then returns the notation or else it continues the whole process from starting and again asks for the starting square chess notation.

```
def create_fen_from_board():
    board = create_board()

    fen = ""

    for row in board:
        empty_count = 0
        for cell in row:
            if cell == '.':
                empty_count += 1
            else:
                if empty_count > 0:
                    fen += str(empty_count)
                    empty_count = 0
                fen += cell

        if empty_count > 0:
            fen += str(empty_count)

        fen += '/'

    fen = fen.rstrip('/')

    fen += ' w - - 0 1'

    return fen
```

Snippet 3.4.17

```
def is_legal_move(fen, move_str):
    board = chess.Board(fen)

    move = chess.Move.from_uci(move_str)

    return move in board.legal_moves
```

Snippet 3.4.18

In the above code a function is defined to check if the move prompted by the player is legal or not. In the function create_fen_board() the board is created using the create_board() function which creates an 8*8 chessboard as a numpy array of strings where each cell has a character representing the piece and its color and a dot for an empty square. The goal of the function is to convert the board position to FEN string **[24]** so that the chess.Board() method can create a board object and then check for legal moves. It then iterates over each row of the board and counts the number of consecutive empty squares squares and appends the count to the FEN string when a non-empty square is encountered. Finally the FEN string is added with the information about the side to move and castling rights. Finally the is_legal_move() checks if the move prompted by the player is legal or not.

```
def recognize_confirmation(r, engine,count):

    model = whisper.load_model("tiny.en")
    positive_words = ['yes', 'yeah', 'correct', 'right', 'yup']
    negative_words = ['no', 'nope', 'incorrect', 'wrong', 'nah']

    positive_pattern = r'\b(?:' + '|'.join(map(re.escape, positive_words)) + r')\b'
    negative_pattern = r'\b(?:' + '|'.join(map(re.escape, negative_words)) + r')\b'


    while True:
        with sr.Microphone() as source:
            print("Please say 'yes' or 'no'")
            engine.say(f"Please say 'yes' or 'no'")
            engine.runAndWait()

            r.adjust_for_ambient_noise(source)

            try:
                audio = r.listen(source, timeout=600, phrase_time_limit=3)
                audio_filename = "captured_audio.wav"

                with open(audio_filename, "wb") as f:
                    f.write(audio.get_wav_data())

                result = model.transcribe(audio_filename)

                text = result["text"]

                positive_matches = re.findall(positive_pattern, text, re.IGNORECASE)
                negative_matches = re.findall(negative_pattern, text, re.IGNORECASE)
```

Snippet 3.4.19

```
                if positive_matches:
                    print("Confirmed")
                    engine.say("Confirmed")
                    engine.runAndWait()
                    return True
                elif negative_matches:
                    print("Not confirmed")
                    engine.say("Not confirmed")
                    engine.runAndWait()
                    return False

            except sr.WaitTimeoutError:
                print("Could not understand audio")
                engine.say("Sorry, I did not understand what you said")
                engine.runAndWait()

            except sr.UnknownValueError:
                print("Could not understand audio")
                engine.say("Sorry, I did not understand what you said")
                engine.runAndWait()

            except sr.RequestError as e:
                print(f"Could not request results from Google Speech Recognition service; {e}")
                engine.say("Sorry, I could not understand what you said")
                engine.runAndWait()
```

Snippet 3.4.20

The above code contains a function recognize_confirmation which is used to confirm the chess notation prompt given by the player. Firstly the whisper AI model is loaded to transcribe the audio confirmation into text. The function enters a loop and continuously listens for confirmation from the player. The adjust_for_ambient_noise method is used to adapt to the surrounding noise. When audio is detected, it is captured and saved to a file named "captured_audio.wav." The loaded language model then transcribes the audio into text using the transcribe method. The transcribed text is processed to identify positive and negative matches based on the defined patterns.

```
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(static_image_mode=True, max_num_hands=2, min_detection_confidence=0.7)
hands_videos = mp_hands.Hands(static_image_mode=False, max_num_hands=2, min_detection_confidence=0.7)
mp_drawing = mp.solutions.drawing_utils

def number_to_letter(number):
    if isinstance(number, str):
        number = int(number)
    return chr(ord('a') + number - 1)

def detectHandsLandmarks(image, hands, draw=True, display = True):
    output_image = image.copy()
    imgRGB = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    results = hands.process(imgRGB)

    if results.multi_hand_landmarks and draw:

        for hand_landmarks in results.multi_hand_landmarks:

            mp_drawing.draw_landmarks(image = output_image, landmark_list = hand_landmarks,
                                      connections = mp_hands.HAND_CONNECTIONS,
                                      landmark_drawing_spec=mp_drawing.DrawingSpec(color=(255,255,255),
                                      thickness=2, circle_radius=2),
                                      connection_drawing_spec=mp_drawing.DrawingSpec(color=(0,255,0),
                                                                                     thickness=2, circle_radius=2))
```

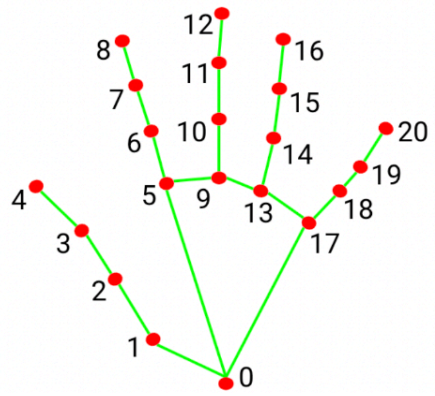
Snippet 3.4.21

```
  if display:

      plt.figure(figsize=[15,15])
      plt.subplot(121);plt.imshow(image[:,:,::-1]);plt.title("Original Image");plt.axis('off');
      plt.subplot(122);plt.imshow(output_image[:,:,::-1]);plt.title("Output");plt.axis('off');

  else:
      return output_image, results
```

Snippet 3.4.22



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

The above code is to detect and visualize hand landmarks of hands using the mediapipe library. MediaPipe Hands is a tool that accurately tracks the position of hands and fingers using machine learning. It can identify 21 key points on the hand in 3D space from a single image. First, it detects the palm of the hand in the picture, and then it calculates the exact position of each finger joint within that area. The above image shows the 21 landmarks that are detected by the mediapipe hand landmark model.

```
def countFingers(image, results, draw=True, display=True):

    height, width, _ = image.shape

    output_image = image.copy()

    count = {'RIGHT': 0, 'LEFT': 0}

    fingers_tips_ids = [mp_hands.HandLandmark.INDEX_FINGER_TIP, mp_hands.HandLandmark.MIDDLE_FINGER_TIP,
                        mp_hands.HandLandmark.RING_FINGER_TIP, mp_hands.HandLandmark.PINKY_TIP]

    fingers_statuses = {'RIGHT_THUMB': False, 'RIGHT_INDEX': False, 'RIGHT_MIDDLE': False, 'RIGHT_RING': False,
                        'RIGHT_PINKY': False, 'LEFT_THUMB': False, 'LEFT_INDEX': False, 'LEFT_MIDDLE': False,
                        'LEFT_RING': False, 'LEFT_PINKY': False}


    for hand_index, hand_info in enumerate(results.multi_handedness):

        hand_label = hand_info.classification[0].label

        hand_landmarks =  results.multi_hand_landmarks[hand_index]
```

Snippet 3.4.23

```
     for tip_index in fingers_tips_ids:

         finger_name = tip_index.name.split("_")[0]

         if (hand_landmarks.landmark[tip_index].y < hand_landmarks.landmark[tip_index - 2].y):

             fingers_statuses[hand_label.upper()+"_"+finger_name] = True

             count[hand_label.upper()] += 1

     thumb_tip_x = hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP].x
     thumb_mcp_x = hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP - 2].x

     if (hand_label=='Right' and (thumb_tip_x < thumb_mcp_x)) or (hand_label=='Left' and (thumb_tip_x > thumb_mcp_x)):

         fingers_statuses[hand_label.upper()+"_THUMB"] = True

         count[hand_label.upper()] += 1
```

Snippet 3.4.24

```
  if draw:

      cv2.putText(output_image, " Total Fingers: ", (10, 25),cv2.FONT_HERSHEY_COMPLEX, 1, (20,255,155), 2)
      cv2.putText(output_image, str(sum(count.values())), (width//2-150,240), cv2.FONT_HERSHEY_SIMPLEX,
                  8.9, (20,255,155), 10, 10)

  if display:

      plt.figure(figsize=[10,10])
      plt.imshow(output_image[:,:,::-1]);plt.title("Output Image");plt.axis('off');

  else:

      return output_image, fingers_statuses, count
```

Snippet 3.4.25

The above function is used for counting the number of fingers that are open. The fingers_tips_ids list contains the landmark IDs for the tips of the index, middle, ring, and pinky fingers and the fingers_statuses dictionary is used to keep track of the status (open or closed) of each finger. For each hand, the function checks the y-coordinate of each finger tip landmark relative to the corresponding landmark of the proximal interphalangeal joint. If the tip is above the joint, it considers the finger to be open, updates the fingers_statuses dictionary accordingly, and increments the finger count for that hand in the count dictionary.

```
def recognizeGestures(image, fingers_statuses, count, draw=True, display=True):
    height, width, _ = image.shape
    output_image = image.copy()
    hands_labels = ['RIGHT', 'LEFT']
    hands_gestures = {'RIGHT': "UNKNOWN", 'LEFT': "UNKNOWN"}

    for hand_index, hand_label in enumerate(hands_labels):
        color = (0, 0, 255)
        if count[hand_label] == 1  and fingers_statuses[hand_label+'_INDEX']:
            hands_gestures[hand_label] = "1"

        elif count[hand_label] == 2  and fingers_statuses[hand_label+'_MIDDLE']
        and fingers_statuses[hand_label+'_INDEX']:
            hands_gestures[hand_label] = "2"

        elif count[hand_label] == 3  and fingers_statuses[hand_label+'_MIDDLE']
        and fingers_statuses[hand_label+'_INDEX'] and fingers_statuses[hand_label+'_RING']:
            hands_gestures[hand_label] = "3"

        elif count[hand_label] == 4 and fingers_statuses[hand_label+'_MIDDLE']
        and fingers_statuses[hand_label+'_INDEX']
        and fingers_statuses[hand_label+'_PINKY'] and fingers_statuses[hand_label+'_RING']:
            hands_gestures[hand_label] = "4"

        elif count[hand_label] == 5:
            hands_gestures[hand_label] = "5"
```

Snippet 3.4.26

```
    elif count[hand_label] == 1 and fingers_statuses[hand_label+'_THUMB']:

        hands_gestures[hand_label] = "6"

    elif count[hand_label] == 3 and fingers_statuses[hand_label+'_INDEX'] and fingers_statuses[hand_label+'_PINKY']
    and fingers_statuses[hand_label+'_MIDDLE']:
        hands_gestures[hand_label] = "7"

    elif count[hand_label] == 3 and fingers_statuses[hand_label+'_INDEX'] and fingers_statuses[hand_label+'_PINKY']
    and fingers_statuses[hand_label+'_THUMB']:
        hands_gestures[hand_label] = "8"

    elif count[hand_label] == 4 and fingers_statuses[hand_label+'_INDEX'] and fingers_statuses[hand_label+'_MIDDLE']
    and fingers_statuses[hand_label+'_THUMB'] and fingers_statuses[hand_label+'_RING']:
        hands_gestures[hand_label] = "9"

    elif count[hand_label] == 2 and fingers_statuses[hand_label+'_INDEX'] and fingers_statuses[hand_label+'_THUMB']
        hands_gestures[hand_label] = "10"

    if draw:

        cv2.putText(output_image, hand_label +': '+ hands_gestures[hand_label] , (10, (hand_index+1) * 60),
                    cv2.FONT_HERSHEY_PLAIN, 4, color, 5)
```

Snippet 3.4.27

```
    if display:

        plt.figure(figsize=[10,10])
        plt.imshow(output_image[:,:,::-1]);plt.title("Output Image");plt.axis('off');

    else:

        return output_image, hands_gestures
```

Snippet 3.4.28

The above function is used for recognizing various hand gestures shown by the user. The function recognizeGestures takes five parameters: image, fingers_statuses, count, draw, and display. image is the input image, fingers_statuses is a dictionary containing the status of each finger, count is a dictionary containing the count of fingers on each hand, draw specifies whether to draw the recognized gestures on the image, and display specifies whether to display the image. The function iterates through each hand label ('RIGHT' and 'LEFT'). Within each iteration, it determines the recognized gesture based on the finger count and finger statuses for that hand. Based on the finger count and statuses, the function assigns a gesture label to each hand. For example, if the finger count is 1 and the index finger is up, it recognizes the gesture as "1". Similarly, for other finger configurations, different gestures are recognized.

```
camera_video = cv2.VideoCapture(0)
camera_video.set(3,1280)
camera_video.set(4,960)

cv2.namedWindow('Gesture', cv2.WINDOW_NORMAL)


def guess():

    counter = 0

    while camera_video.isOpened():

        num_of_frames = 5

        ok, frame = camera_video.read()

        if not ok:
            continue

        frame = cv2.flip(frame, 1)

        frame, results = detectHandsLandmarks(frame, hands_videos, display=False)
```

Snippet 3.4.29

```
    if results.multi_hand_landmarks:

        frame, fingers_statuses, count = countFingers(frame, results, draw=False, display=False)


        frame, hands_gestures = recognizeGestures(frame, fingers_statuses, count, draw = False, display=False)
        print(hands_gestures)

        if all(hand_gesture != "UNKNOWN" for hand_gesture in hands_gestures.values()):

            if counter==0:
                left_hand_sign = hands_gestures['LEFT']
                right_hand_sign = hands_gestures['RIGHT']
                counter+=1

            elif (hands_gestures['LEFT'], hands_gestures['RIGHT']) == (left_hand_sign, right_hand_sign):
                counter += 1

                if counter == num_of_frames:

                    return (number_to_letter(left_hand_sign)+right_hand_sign)

            else:
                counter = 0
```

Snippet 3.4.30

```
          else:

              counter = 0


      cv2.imshow('Gesture', frame)

      k = cv2.waitKey(1) & 0xFF

      if(k == 27):
          break

  camera_video.release()
  cv2.destroyAllWindows()
```

Snippet 3.4.31

The above code initializes a video capture object camera_video using OpenCV's VideoCapture function. It sets the capture device index to 0 (default webcam) and specifies the desired frame dimensions as 1280x960 pixels using the set method. It creates a resizable window named 'Gesture' using OpenCV's namedWindow function. The guess function enters a loop that continuously captures frames from the webcam while the video capture object is open. It reads a frame from the webcam using the read method of the video capture object. The frame is stored in the frame variable. If hand landmarks are detected (results.multi_hand_landmarks), the countFingers and recognizeGestures functions are called to count the fingers and recognize gestures for each hand, respectively. The recognized gestures are stored in the hands_gestures dictionary. If all recognized gestures are not "UNKNOWN" (i.e., meaningful gestures are detected), the function checks if the same gestures persist for five consecutive frames. If the gestures remain the same for five continuous frames then it returns the recognized gestures as a combination of letters representing the left and right hand gestures.

```
from chessdotcom import get_player_games_by_month_pgn, Client
import datetime

Client.request_config["headers"]["User-Agent"] = (
    "My Python Application. "
    "Contact me at email@example.com"
)

username = "Happy02001"

year = 2024
month = 3

response = get_player_games_by_month_pgn(username, year=year, month=month)

pgn_content = response.json['pgn']['pgn']

filename = f"ChessCom_{username}_{year}{month:02d}.pgn"

headers = {
    "Content-Type": "application/x-chess-pgn",
    "Content-Disposition": f'attachment; filename="{filename}"'
}
```

Snippet 3.4.32

```
pgn_content_cleaned = ""
inside_clock = False
for char in pgn_content:
    if char == '[':
        inside_clock = True
    elif char == ']':
        inside_clock = False
    elif not inside_clock and char != '.' and char != '{' and char != '}':
        pgn_content_cleaned += char

games = pgn_content_cleaned.split("\n\n\n")
```

Snippet 3.4.33

```
games = pgn_content_cleaned.split("\n\n\n")



for idx, game in enumerate(games, start=1):
    if not game.strip():
        continue

    moves = game.split()

    cleaned_moves = []
    prev_digit = ""

    for move in moves:
        if move.isdigit():
            if move!=prev_digit:
                cleaned_moves.append(move+".")
                prev_digit = move
        else:
            cleaned_moves.append(move)

    cleaned_game = " ".join(cleaned_moves)

    print(cleaned_game)
    print("-" * 50)
```

Snippet 3.4.34

The above code is used to retrieve the games played by a specific chess player from the website chess.com for a given month and year. The code sends an API request using the get_player_games_by_month_pgn function to fetch the games of the specified player for the given month and year. The response is stored in the response variable. The PGN content is cleaned to remove unnecessary characters like clock times and comments and duplicate numbers. This is done by iterating through each character in the PGN content and appending only relevant characters to pgn_content_cleaned.

```
import requests
import json

def fetch_user_games(username):
    url = f"https://lichess.org/api/games/user/{username}?literature=true"
    headers = {
        "Accept": "application/x-ndjson"
    }

    try:
        response = requests.get(url, headers=headers)
        response.raise_for_status()
        games_data = []
        for line in response.iter_lines():
            game = json.loads(line)
            games_data.append(game)
        return games_data
    except requests.exceptions.RequestException as e:
        print("Error fetching user games:", e)

username = "Happy02001"
user_games = fetch_user_games(username)
print(user_games)
```

Snippet 3.4.35

The above code is used to retrieve all the games played by a specific chess player from the Lichess website. The code sends an API request using the fetch_user_games function that takes a username parameter to fetch all the games of the specified player. We construct the URL to fetch the games for the given username. The username is inserted into the URL using an f-string. A dictionary named headers is defined which contains the HTTP header settings. Specifically, we set the Accept header to indicate that we are expecting the response in ndjson format. A GET request is sent to the specified URL with the custom headers and we get the PGN data in response. Additionally, the code employs error handling with the response.raise_for_status() method to manage potential HTTP request errors effectively.

```
import requests
import chess


def fetch_daily_puzzle():
    url = "https://lichess.org/api/puzzle/daily"
    headers = {
        "Accept": "application/json"
    }

    try:
        response = requests.get(url, headers=headers)
        response.raise_for_status()
        puzzle_data = response.json()

        pgn_string = puzzle_data["game"]["pgn"]
        theme = puzzle_data["puzzle"]["themes"]
        color_to_move = puzzle_data["game"]["players"][0]["color"]

        return pgn_string, theme, color_to_move

    except requests.exceptions.RequestException as e:
        print("Error fetching puzzle:", e)
```

Snippet 3.4.36

```
def pgn_to_board(pgn_string):
    moves = pgn_string.split()
    board = chess.Board()
    for move in moves:
        board.push_san(move)
    return board



pgn_string, theme, color_to_move = fetch_daily_puzzle()

board = pgn_to_board(pgn_string)

print(board)
```

Snippet 3.4.37

The above code contains two functions: fetch_daily_puzzle and pgn_to_board. These functions work together to fetch the daily chess puzzle from the Lichess API, convert the puzzle's PGN (Portable Game Notation) string to a chess board representation, and then print the resulting board. The fetch_daily_puzzle function sends an HTTP GET request to the Lichess API endpoint for fetching the daily puzzle. The URL for the API endpoint is "https://lichess.org/api/puzzle/daily". The request includes headers specifying that the response should be in JSON format. If the request is successful (status code 200), the function parses the JSON response and extracts relevant puzzle data such as the PGN string. The pgn_to_board function takes a PGN string representing a chess game as input. It splits the PGN string into individual moves using the split method, initializes an empty chess board object using chess.Board() and iterates through each move in the PGN string and pushes the move to the board using the push_san method.

```
from stockfish import Stockfish
stockfish = Stockfish(path=r"C:\Users\R.P.JOSHI\Desktop\Blind mate chess all work till now\Blind Mate Chess with audio
(latest version)\stockfish\stockfish-windows-x86-64-avx2.exe")

def get_best_move():
    return stockfish.get_best_move()
def get_evaluation():
    return stockfish.get_evaluation()
def get_n_moves(n):
    return stockfish.get_top_moves(n)
def stats():
    return stockfish.get_wdl_stats()
stockfish.set_fen_position("r4kn1/2p5/4pp2/1p2n3/p3P3/3PQ3/B1q4r/K5R1 b - - 1 25")
print("Initial board position\n", stockfish.get_board_visual())

best_move = get_best_move()
before_eval = get_evaluation()
print("Evaluation before move:", before_eval)
move = input("\nEnter your move (in UCI notation, e.g., e2e4): ")
stockfish.make_moves_from_current_position([move])
print("\nBoard Position after suggested move\n", stockfish.get_board_visual())
after_eval = get_evaluation()
print("Evaluation after move:", after_eval)
print("Best Move:", best_move,"\n\n")
n = int(input("Top (n) moves you want to get: ", ))
print(get_n_moves(n))
```

Snippet 3.4.38

The above code uses the stockfish chess engine library to perform evaluation of a chess position. The get_best_move() function retrieves the best move suggested by Stockfish for the current position, get_evaluation()function retrieves the evaluation score of the current position by Stockfish, get_n_moves(n) function retrieves the top n moves suggested by Stockfish for the current position, stats() function retrieves statistics about the current position, such as win/draw/loss probabilities. It sets up an initial chess position using the set_fen_position() method, which accepts a FEN (Forsyth–Edwards Notation) string representing the board position. The code asks for a move from the user and prints the evaluation of the position before and after the move by the user.

```
import io
import chess.pgn

pgn_moves = "1. e4 e5 2. d3 Nc6 3. Nc3 Nf6 4. Nf3 d5 5. exd5 Nxd5 6. Nxd5 Qxd5"

board = chess.Board()

pgn = io.StringIO(pgn_moves)
game = chess.pgn.read_game(pgn)
board = game.board()

fen = board.fen()

print("\nFEN notation after the moves:")
print(fen,"\n")
```

Snippet 3.4.39

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from stockfish import Stockfish
import chess
import chess.engine
import math

engine = chess.engine.SimpleEngine.popen_uci("C:/Users/cbarg/Downloads/
stockfish_14_win_x64_avx2/stockfish_14_win_x64_avx2/stockfish_14_x64_avx2.exe")
stockfish = Stockfish('C:/Users/cbarg/Downloads/stockfish_14_win_x64_avx2/
stockfish_14_win_x64_avx2/stockfish_14_x64_avx2.exe')
stockfish.set_elo_rating(2600)
df = pd.read_csv('chess_games.csv')
```

Snippet 3.4.40

The above code sets up an environment for analyzing and processing data related to chess games. It imports essential libraries such as numpy, pandas, matplotlib, and seaborn for data analysis and visualization. Additionally, it imports the Stockfish engine and chess modules, enabling interaction with chess engines and performing chess-related tasks programmatically. The Stockfish engine is instantiated twice, once using the Stockfish class and once using the chess.engine.SimpleEngine.popen_uci method, each providing the path to the Stockfish executable. The Elo rating of the Stockfish engine is set to 2600. Furthermore, a CSV file named 'chess_games.csv' is read into a pandas DataFrame named df, containing data related to chess games for further analysis.

```
print(df.dtypes)
df.head(1)
```

Snippet 3.4.41

The above code outputs the data types of each column in the DataFrame df. This gives a quick overview of the data types present in the DataFrame, which is helpful for understanding the nature of the data being worked with. It also displays the first row of the DataFrame df by using the head() method with an argument of 1, only the first row is returned. This is useful for inspecting the structure and content of the DataFrame, providing a glimpse into the actual data values present in the dataset.

```
df = df[df.turns >= 3]

df['moves_list'] = df.moves.apply(lambda x: x.split())

df['opening_move'] = df.moves_list.apply(lambda x: x[0])
df['response'] = df.moves_list.apply(lambda x: x[1])

df['opening_name'] = df.moves_list.apply(lambda x: 'King\'s Pawn' if x[0] == 'e4' else ('Queen\'s Pawn' if x[0] == 'd4' else
 ('English' if x[0] == 'c4' else ('Reti' if x[0] == 'Nf3' else 'Other'))))

df['rating_gap'] = abs(df['white_rating'] - df['black_rating'])
df['higher_rated_victory'] = np.where((df['winner'] == 'White') & (df['white_rating'] > df['black_rating']) | (df['winner'] == 'Black')
& (df['black_rating'] > df['white_rating']), 1, 0)
df['rating_gap_class'] = pd.cut(df.rating_gap, bins=[0, 50, 100, 150, 200, 250, np.inf],
                                labels=['0-50', '51-100', '101-150', '151-200', '201-250', '>250'])

df['white_victory'] = np.where(df['winner'] == 'White', 1, 0)

opening_data = df.groupby('opening_name')['game_id'].count()

plt.pie(x=opening_data, autopct='%.1f%%', labels=opening_data.keys(), pctdistance=0.5)
plt.title('Percentage of Openings', fontsize=14)
plt.axis('equal')
plt.show()
plt.clf()
```

Snippet 3.4.42

```
opening_text = " ".join([f"{name}: {percent:.1f} percent."
            for name, percent in zip(opening_data.keys(), opening_data.values)])

engine = pyttsx3.init()

engine.setProperty('rate', 150)

engine.say(opening_text)
engine.runAndWait()
```

Snippet 3.4.43

The code filters the DataFrame df to include only games with at least 3 turns using the line df = df[df.turns >= 3]. This ensures that only meaningful games are considered for analysis. The moves column is split into a list of moves using df['moves_list'] = df.moves.apply(lambda x: x.split()). This allows for easier extraction of opening moves and responses. The opening move and response for each game are extracted and stored in separate columns using lambda functions. Then the opening moves are categorised into opening names such as "King's Pawn," "Queen's Pawn," "English," "Reti," or "Other" based on the first move played by each player. Following that rating gap between players is calculated and it is determined whether the higher-rated player won the game. These calculations are stored in columns named rating_gap, higher_rated_victory, and rating_gap_class. The games are grouped by opening name and the number of games for each opening is counted using groupby. The pyttsx3 library is used to initialize a text-to-speech engine, and the opening percentages are spoken out loud using this engine.

```
english_winner_data = english.groupby('winner')['game_id'].count()
queen_winner_data = queens_pawn.groupby('winner')['game_id'].count()
king_winner_data = kings_pawn.groupby('winner')['game_id'].count()
reti_winner_data = reti.groupby('winner')['game_id'].count()

pie, axs = plt.subplots(2,2, figsize=[10,10])

plt.subplot(2,2,1)
plt.pie(x=english_winner_data, autopct="%.1f%%", labels=english_winner_data.keys(), pctdistance=0.5)
plt.title("Winners After a c4 Opening", fontsize=14)
plt.axis('equal')

plt.subplot(2,2,2)
plt.pie(x=queen_winner_data, autopct="%.1f%%", labels=queen_winner_data.keys(), pctdistance=0.5)
plt.title("Winners After a d4 Opening", fontsize=14)
plt.axis('equal')

plt.subplot(2,2,3)
plt.pie(x=king_winner_data, autopct="%.1f%%", labels=king_winner_data.keys(), pctdistance=0.5)
plt.title("Winners After a e4 Opening", fontsize=14)
plt.axis('equal')

plt.subplot(2,2,4)
plt.pie(x=reti_winner_data, autopct="%.1f%%", labels=reti_winner_data.keys(), pctdistance=0.5)
plt.title("Winners After a Nf3 Opening", fontsize=14)
plt.axis('equal')
plt.show()
```

Snippet 3.4.44

```
engine = pyttsx3.init()

def speak_winning_percentages(opening, percentage):
    engine.say(f"In the {opening} opening, the winning percentage is {percentage:.1f} percent.")

for opening, data in zip(["c4", "d4", "e4", "Nf3"], [english_winner_data, queen_winner_data, king_winner_data, reti_winner_data]):
    total_games = sum(data)
    if total_games > 0:
        percentage = (data['white'] / total_games) * 100
        speak_winning_percentages(opening, percentage)

engine.runAndWait()
```

Snippet 3.4.45

The above code uses the matplotlib library to create four pie charts, each representing the winning percentages for a specific chess opening. The openings analyzed are the English (after the move c4), Queen's Pawn (after d4), King's Pawn (after e4), and Reti (after Nf3). The code computes the winning percentages for both White and Black, as well as draw percentages for each opening based on the game data grouped by the winner. These winning percentages are then visualized using pie charts, with each chart representing one opening. Finally pyttsx3 library is used to convey the winning percentages for each opening to the blind player.

```
df_grouped_ratings = df.groupby('opening_name')['higher_rated_victory'].sum()
df_grouped_ratings = df_grouped_ratings.to_frame()
df_grouped_ratings['totals'] = df.groupby('opening_name')['higher_rated_victory'].count()
df_grouped_ratings['losses_or_draws'] = df_grouped_ratings['totals'] - df_grouped_ratings['higher_rated_victory']
print(df_grouped_ratings.head(10))

df_grouped_ratings = df_grouped_ratings.sort_values('totals', ascending=False)
fig, ax = plt.subplots(1, figsize=(12,10))
ax.bar([0,1,2,3,4], df_grouped_ratings['higher_rated_victory'],
       label='Higher Rating Win', color='#ae24d1', tick_label=df_grouped_ratings.index)
ax.bar([0,1,2,3,4], df_grouped_ratings['losses_or_draws'],
       label='Draw or Lower Rating Win', bottom=df_grouped_ratings['higher_rated_victory'], color='#24b1d1')
ax.set_ylabel('Wins', fontsize=14)
ax.set_xlabel('Opening Move', fontsize=14)
ax.set_title('Wins by Opening Move', fontsize=18)
ax.legend()
plt.show()


engine = pyttsx3.init()

def speak_wins(opening, wins):
    engine.say(f"In the {opening} opening, the higher-rated player wins {wins} times.")

for opening, wins in zip(df_grouped_ratings.index, df_grouped_ratings['higher_rated_victory']):
    speak_wins(opening, wins)

engine.runAndWait()
```

Snippet 3.4.46

The above code is used to analyze the frequency of wins for the higher-rated player along with draw or win for lower rated player across different chess opening moves. It starts by grouping the DataFrame by the 'opening_name' column and summing the occurrences where the higher-rated player wins ('higher_rated_victory'). These values are then stored in a DataFrame called 'df_grouped_ratings'. Additionally, the total number of games played for each opening move is calculated and stored in a new column named 'totals'. Another column, 'losses_or_draws', is computed to represent the number of games where the higher-rated player did not win (i.e., either lost or drew). The code then prints the first 10 rows of the DataFrame to inspect the data. After that, the DataFrame is sorted based on the total number of games played ('totals') in descending order. This sorted DataFrame is used to create a bar plot visualizing the wins by opening move. The higher-rated player's wins are represented in one color ('#ae24d1'), while the games where the higher-rated player did not win (either lost or drew) are shown in another color ('#24b1d1'). The x-axis represents different opening moves, and the y-axis indicates the number of wins. A legend is included to distinguish between wins and losses/draws. Text-to-speech functionality is also used to inform the player about the statistics.

```
engine = pyttsx3.init()

def speak_statistics(title, xlabel, ylabel, data):
    engine.say(f"In the {title} subplot, the white winning rate is shown based on Black's response.")
    engine.say(f"The x-axis represents Black's response, and the y-axis represents the white winning rate.")
    for index, row in data.iterrows():
        engine.say(f"When Black responds with {row['response']}, the white winning rate is {row['white_victory']:.2f}.")

plt.clf()
pie, axs = plt.subplots(2,2, figsize=[16,10])

plt.subplot(2,2,1)
sns.barplot(
    data=kings_pawn,
    x='response',
    y='white_victory',
)
plt.title('King\'s Pawn Opening Response')
plt.ylabel('White Winning Rate')
plt.xlabel('Black\'s Response')
plt.axhline(0.5)
speak_statistics("King's Pawn", "Black's Response", "White Winning Rate", kings_pawn)
```

Snippet 3.4.47

```
plt.subplot(2,2,2)
sns.barplot(
    data=queens_pawn,
    x='response',
    y='white_victory',
    palette='Spectral'
)
plt.title('Queen\'s Pawn Opening Response')
plt.ylabel('White Winning Rate')
plt.xlabel('Black\'s Response')
plt.axhline(0.5)
speak_statistics("Queen's Pawn", "Black's Response", "White Winning Rate", queens_pawn)

plt.subplot(2,2,3)
sns.barplot(
    data=english,
    x='response',
    y='white_victory',
    palette='brg_r'
)
plt.title('English Opening Response')
plt.ylabel('White Winning Rate')
plt.xlabel('Black\'s Response')
plt.axhline(0.5)
speak_statistics("English", "Black's Response", "White Winning Rate", english)
```

Snippet 3.4.48

```
plt.subplot(2,2,4)
sns.barplot(
    data=reti,
    x='response',
    y='white_victory',
    palette='Wistia'
)
plt.title('Reti Opening Response')
plt.ylabel('White Winning Rate')
plt.xlabel('Black\'s Response')
plt.axhline(0.5)
speak_statistics("Reti", "Black's Response", "White Winning Rate", reti)

plt.show()
plt.clf()

engine.runAndWait()
```

Snippet 3.4.49

The above code aims to visualize and audibly convey the winning rates of White in response to different moves made by Black across four different chess opening scenarios. It begins by creating a figure with four subplots, each representing a specific chess opening scenario: King's Pawn, Queen's Pawn, English, and Reti. Within each subplot, a seaborn bar plot is generated, where the x-axis denotes the moves made by Black in response to the opening move, and the y-axis represents the winning rate of White in percentage. A horizontal line is drawn at y=0.5 to indicate the 50% winning rate threshold. The speak_statistics function is defined to articulate the statistics for each subplot. It narrates the title of the subplot, the axes labels, and iterates through the data to verbally convey the winning rate of White corresponding to each move made by Black.

```
def speak_statistics(df):
    engine.say("In games where both players have a rating under {} ELO:".format(num))
    for winner, count in df.items():
        engine.say(f"The percentage of games won by {winner} is {count / df.sum() * 100:.1f} percent.")

num = 1100
df_under_num = df[(df.white_rating < num) & (df.black_rating < num)]
df_under_num_winners = df_under_num.groupby('winner')['game_id'].count()

plt.pie(x=df_under_num_winners, autopct="%.1f%%", labels=df_under_num_winners.keys(), pctdistance=0.5)
plt.title("Winning Percentage for Games Under {} ELO".format(num), fontsize=14)
plt.axis('equal')
plt.show()

speak_statistics(df_under_num_winners)

engine.runAndWait()
```

Snippet 3.4.50

The above code aims to visualize and audibly convey the winning percentages of games where both players have a rating under a certain threshold (ELO) through a pie chart and text-to-speech (TTS) functionality. It begins by filtering the DataFrame to include only games where both players' ratings are below the specified ELO threshold. Then, it calculates the winning percentages for each player category (White, Black, or Draw) based on the filtered DataFrame. The pie chart is generated to visually represent these winning percentages. Simultaneously, the TTS engine is used to verbally convey the statistics, iterating through the DataFrame to narrate the percentage of games won by each player category.

```
def speak_statistics(df):
    engine.say("The win rate by rating gap is as follows:")
    for index, row in df.iterrows():
        engine.say(f"For a rating gap of {row['rating_gap_class']},
        the win rate by higher ratings is {row['higher_rated_victory']:.2f} percent.")

sns.barplot(
    data=df,
    x='rating_gap_class',
    y='higher_rated_victory',
    order=['0-50', '51-100', '101-150', '151-200', '201-250', '>250'],
)
plt.xlabel('Rating Gap', fontsize=16)
plt.ylabel('Win Rate by Higher Ratings', fontsize=16)
plt.title('Win Rate by Rating Gap', fontsize=16)
plt.show()
plt.clf()
speak_statistics(df)
engine.runAndWait()
```

Snippet 3.4.51

The above code makes use of Seaborn to visualize how the rating gap between players affects the outcome of chess games, particularly focusing on the win rate by higher ratings. It creates a bar plot where the x-axis represents different rating gap classes, ordered from smallest to largest, and the y-axis indicates the win rate by higher-rated players. Text-to-speech (TTS) functionality is used to for visually impaired users, with a function defined to iterate through the DataFrame and verbally convey the win rate for each rating gap class.

3.5 Challenges

1. Accurately identifying user prompt:

One of the key challenges in the project was to accurately identify the chess move notations and other prompts given by the user. Initially we made the use of Python's SpeechRecognition library to transcribe the spoken words into text but it resulted in frequent errors and misunderstandings. To overcome this problem we explored an alternative open source speech recognition model provided by Open AI's Whisper AI model. After the integration of this model into the application there was a significant improvement in the accuracy of speech recognition. We then also implemented a hand gesture recognition model in the application to make the process of prompt input more effective.

2. Efficient chessboard detection:

Another challenge faced during the project was efficient detection of chess boards on the computer screen. The initial approach for detecting the chess board involved using a YOLO(you only look once) object detection model to locate the chess board. Although the model was successful in detecting the chessboard, its computational demands resulted in a significant amount of delay in chessboard detection. To overcome this problem we made the use of image recognition techniques. According to a research paper on comparison of edge detection techniques, sobel filters for edge detection were found to be more simple and less time consuming than canny edge detector **[25]**. With the help of Sobel operator and HoughLines algorithm we were able to efficiently detect lines in the image and among those detected lines we looked for 9 equally spaced horizontal and vertical lines amongst the detected lines to accurately identify the position of the chessboard on the screen.

3. Real time monitoring of game moves and game status:

Another key challenge in the project was to monitor the move of the opponent in real time and also get the game status in real time. The initial approach we took to handle this problem was to at regular intervals in the game get the current position of pieces on the board by applying the machine learning model on the chessboard to identify the chess piece on each square and them compare the position with the previously calculated position to see if there is any change in the position. This approach will be a bit time consuming and also since chess games have time control in them the player cannot afford to lose even one second in the game. Therefore to handle this problem we take the help of Lichess API to stream game moves of the player and as soon as the opponent plays its move the API returns the move played by the opponent and the current status of the game.

Chapter-04 Testing

4.1 Testing Strategy

Unit Testing:

In Unit testing we focussed on individual components of the code. The project code consists of various modules in it such as Chessboard detection, Piece recognition, Lichess API integration, Hand gesture recognition etc. The unit testing strategy focussed on validating the functionalities of these modules independent of each other and ensuring that each one of them perform their tasks accurately and efficiently. Firstly unit testing was done to evaluate the chessboard detection and pieces identification. Different chessboard images with different styles and colors and also of different sizes were taken to test on them. The main objective of it was to verify if the detected board size and coordinates matched with the expected values and also if the model was able to accurately identify different chess pieces under various conditions. Speech recognition model was thoroughly tested by different individuals and under different ambient noise conditions to check if prompt by the user was accurately identified or not. Move generation functions were tested using user inputs to check if the start and end coordinates were aligned with the square notations or not. Liches API testing was also done for different scenarios including draw, resignation and checkmate. Hand gesture recognition model's testing was done by inputting various images containing different hand gestures in different lighting conditions, camera angles and hand orientations to include all possible scenarios.

End To End Testing:

In end to end testing we tested the entire application from start to finish. This helped in verifying whether the application works fine as a whole or not. It started with opening the Lichess Chess platform and starting a game with random players. Once the game started the application was launched and firstly the application identified the chessboard on the screen and its coordinates. Then it recognized each and every piece on the board and saved the current position of the chessboard. Next we tested it for user interaction using speech recognition and hand gesture recognition. While playing different games and giving different commands to the application we verified the applications response to various game events such as checkmates, resignation or draws. The testing approach mimicked real world scenarios that could possibly occur in the game and made sure the application responds to it in the desired way. Simulation of chess moves was thoroughly tested by prompting various kinds of legal and non legal moves in the game. In case of a legal move the application was tested to make sure that it makes accurate moves on the board. The testing strategy also included scenarios such as network issues and incorrect voice commands.

4.2 Test Cases and Outcomes

Test Case 1: Chessboard detection

Objective: To verify that the chessboard detection components of the application can correctly identify the chessboard.

Input: Different screenshots of the chess boards with different sizes, different styles with all pieces in random positions were given as input to the function.

Expected Outcome: The application should be able to correctly identify straight lines in the screenshot and also calculate the coordinates of the chessboard.

Actual Outcome: The application correctly identified all the straight lines in the screenshot and accurately filtered the lines that were at equal distance and also calculated the coordinates of those lines because of which the chessboard was detected successfully.

Test Case 2: Pieces Detection

Objective: To verify that the YOLO model correctly identifies all the chess pieces

Input: A screenshot of a chessboard containing different pieces of different styles and backgrounds in different positions.

Expected Outcome: The application should be able to correctly identify the type and the color of each chess piece.

Actual Outcome: The application correctly identified the type of chess piece and its color almost in all cases.

Test Case 3: Speech Recognition and Gesture Recognition

Objective: To verify that the speech recognition and gesture recognition components of the application can correctly recognize move notations and prompts from the player.

Input: A set of live audio recordings of the player saying different move notations and prompts along with images of the person with various gestures was given as input

Expected Outcome: The application should correctly transcribe each audio recording and the gesture and identify accurately the prompt of the user

Actual Outcome: The application correctly transcribed almost all audio recordings and gestures except a few who had a bit more ambient noise and at different angles respectively.

Chapter-05 Results and Evaluation

5.1 Results

The blind mate chess project has been successfully developed and tested. The application is able to correctly identify the chessboard and pieces, recognize move notations and prompts from the player, and play a game of chess against a human opponent.

1. Chessboard Detection:

- The Sobel filter accurately highlighted edges, where there was change in gradient magnitude.

- The Hough transform accurately identified vertical lines, helping in the accurate detection of chessboard boundaries.

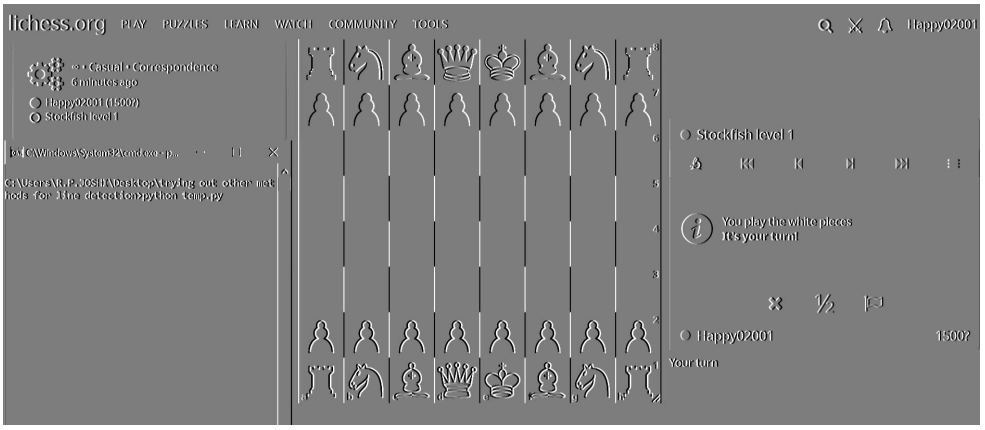


Figure 5.1 Sobel Image of chessboard

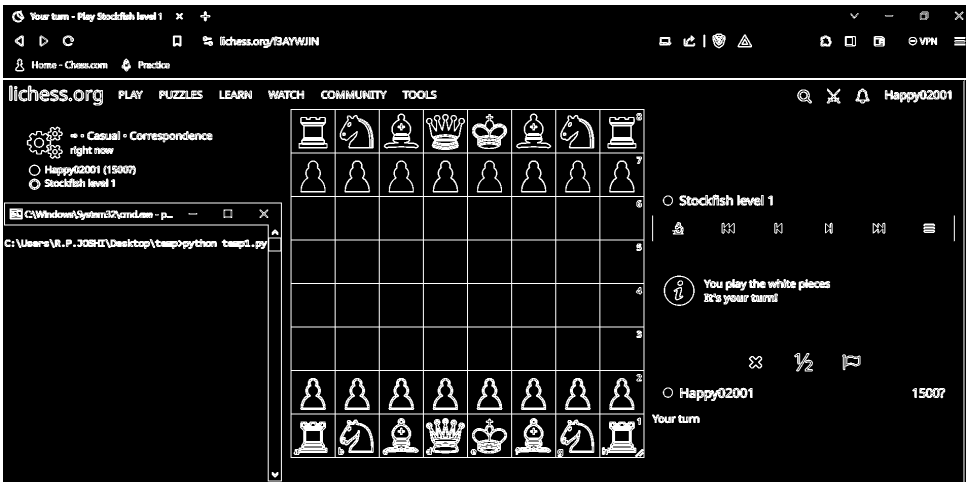


Figure 5.2 Binary edge map of chessboard

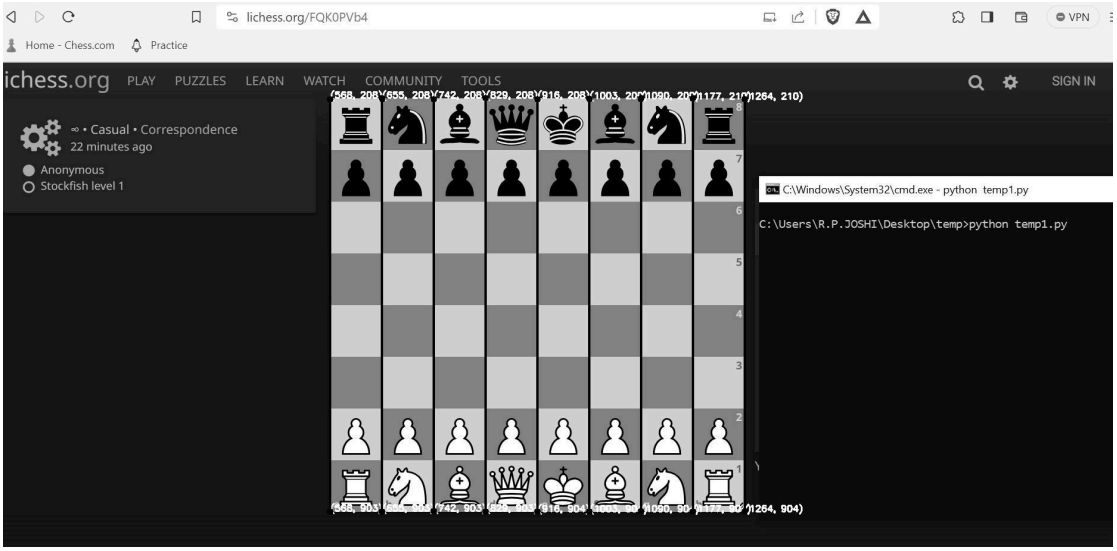


Figure 5.3 Chessboard detected lines with their coordinates

- During chessboard detection two sets of images were tested. In the first test there were clear images of chess boards and in the second test images were blurry and occluded. The following table shows the performance of the algorithm on both sets of images.

TABLE I

EVALUATION METRICS FOR CHESSBOARD DETECTION

Dataset	*Total Images*	*Correctly Detected*	*Incorrectly Detected*	*Accuracy*	*Mean Pixel Deviation*
Test A	160	155	5	96.88%	1.3 pixels
Test B	40	37	3	92.50%	1.8 pixels
Overall	200	192	8	96.00%	1.5 pixels

2. Piece Recognition:

- The YOLO model for the piece recognition consistently identified the type and color of various chess pieces, and helped in providing an accurate board representation. The following table shows the performance of the YOLO model on both sets of images.

Figure 5.4 Recognized chess piece with their confidence score

TABLE II

EVALUATION METRICS FOR CHESS PIECES DETECTION

Chess Piece	*Test Set*	*Total Images*	*Correctly Detected*	*Incorrectly Detected*	*Accuracy*
Pawn	A	40	38	2	95.00%
Rook	A	40	37	3	92.50%
Knight	A	40	39	1	97.50%
Bishop	A	40	38	2	95.00%
King	A	40	38	2	95.00%
Queen	A	40	38	2	95.00%
Pawn	B	10	9	1	90.00%
Rook	B	10	9	1	90.00%
Knight	B	10	9	1	90.00%
Bishop	B	10	9	1	90.00%
King	B	10	9	1	90.00%
Queen	B	10	9	1	90.00%
Overall	-	250	230	20	92.00%

3. Gesture Recognition:

- The mediapipe framework was accurately able to track the hands and the hand gesture recognition algorithm was able to successfully identify and classify hand gestures in real-time video input.

TABLE III

EVALUATION METRICS FOR GESTURE RECOGNITION

Gesture	*Total Images*	*Correctly Detected*	*Incorrectly Detected*	*Accuracy*
One	25	24	1	96.00%
Two	25	24	1	96.00%
Three	25	24	1	96.00%
Four	25	24	1	96.00%
Five	25	24	1	96.00%
Six	25	23	2	96.00%
Seven	25	23	2	92.00%
Eight	25	22	3	88.00%
Yes	25	23	2	92.00%
No	25	22	3	88.00%
Overall	250	231	19	92.40%

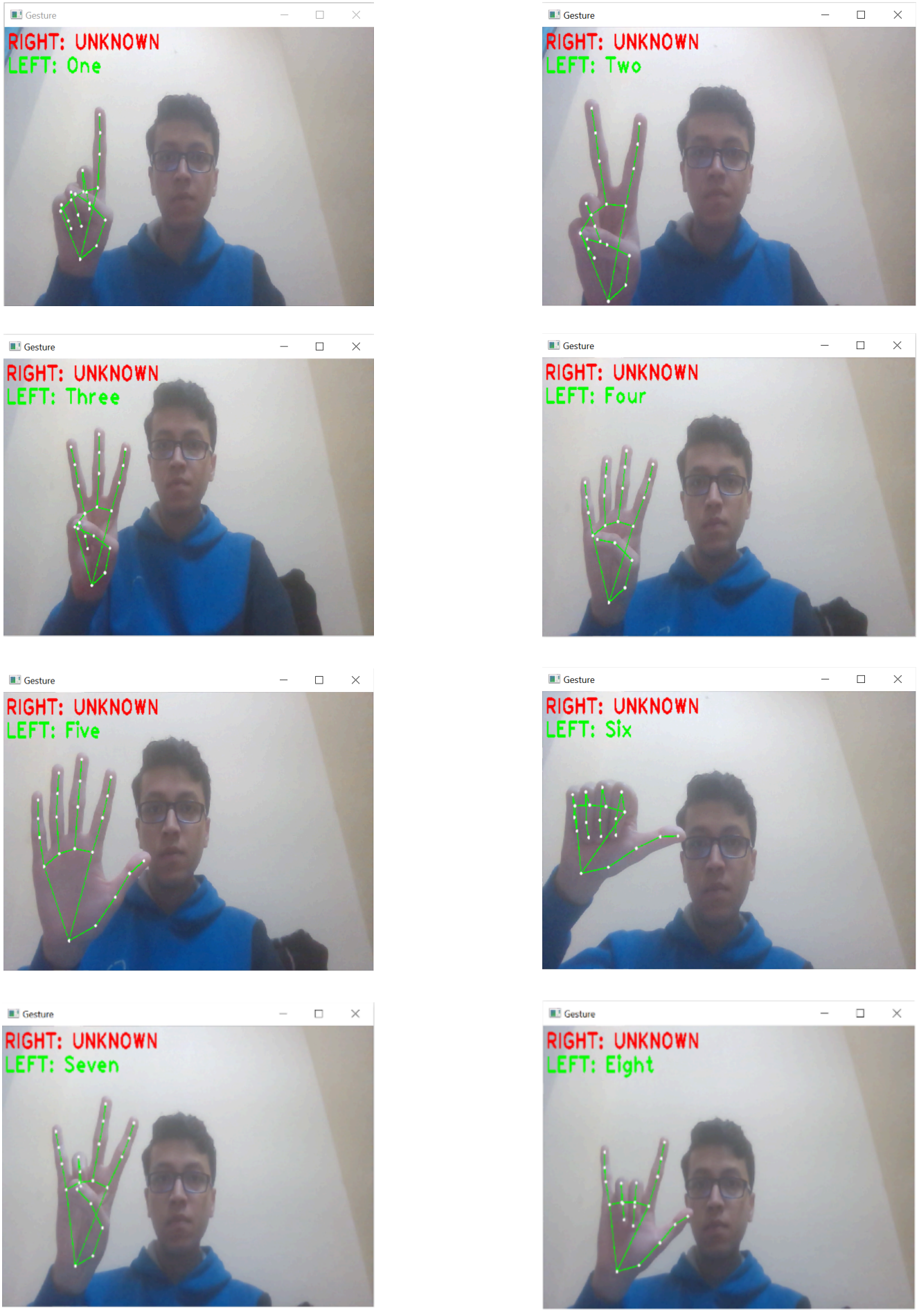


Figure 5.5 Different gestures recognized by the model

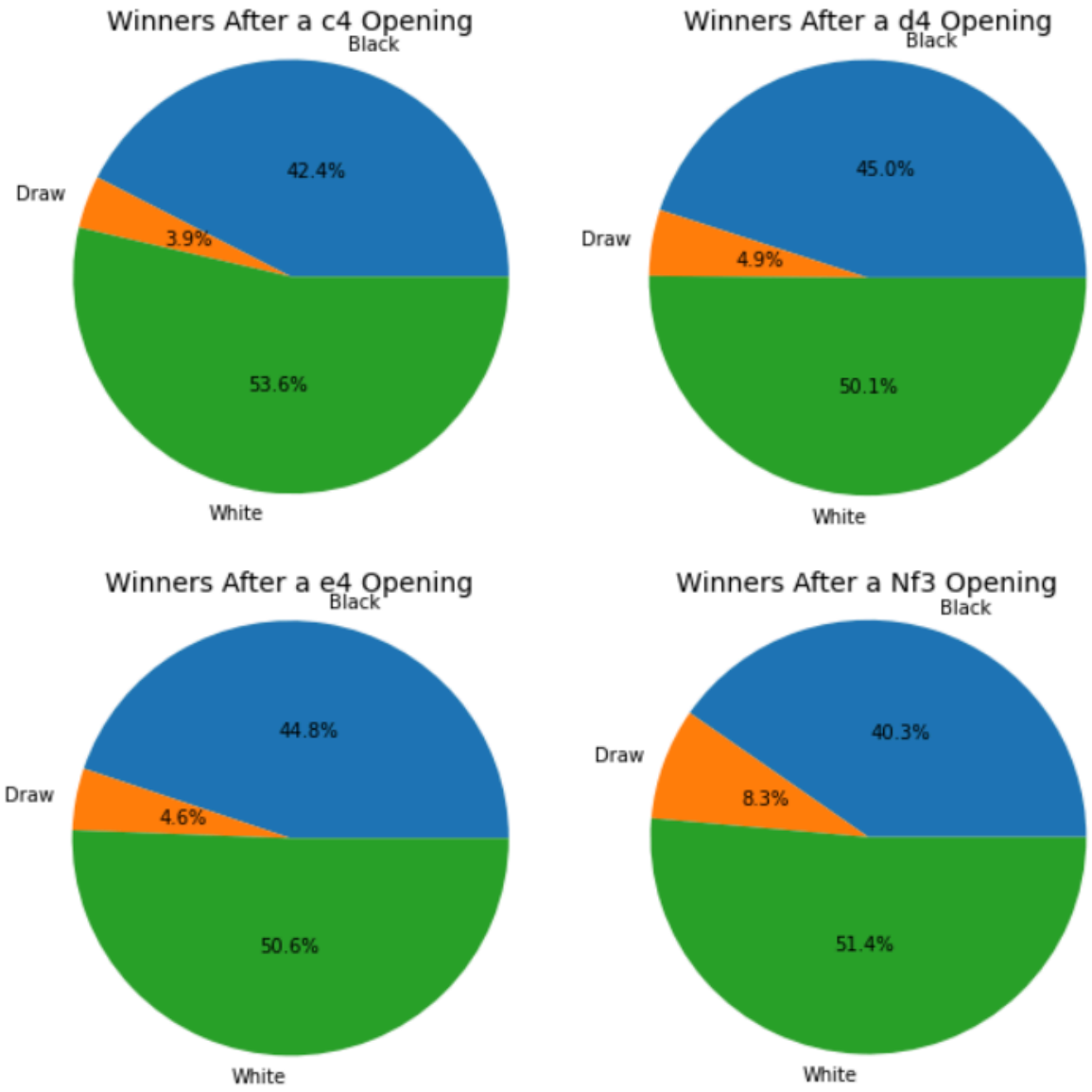


Figure 5.6 Win percentage after a particular opening

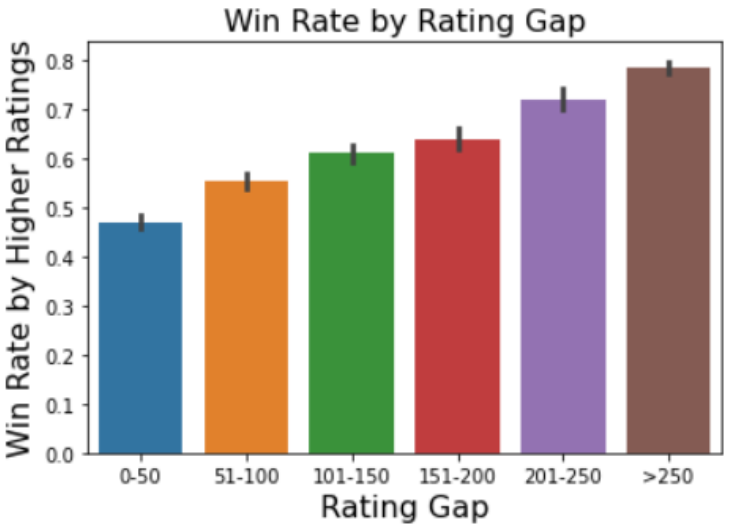


Figure 5.7 Win probability by rating gap

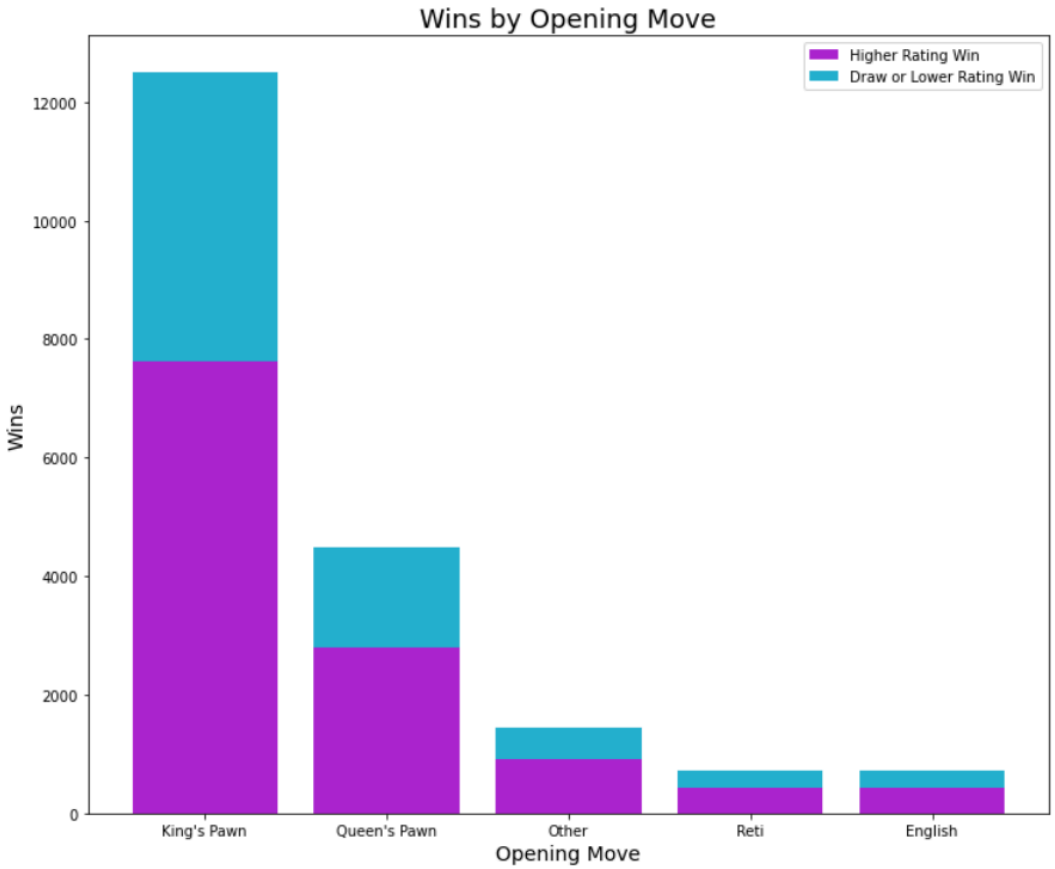


Figure 5.8 Games won against higher and lower rated players by opening

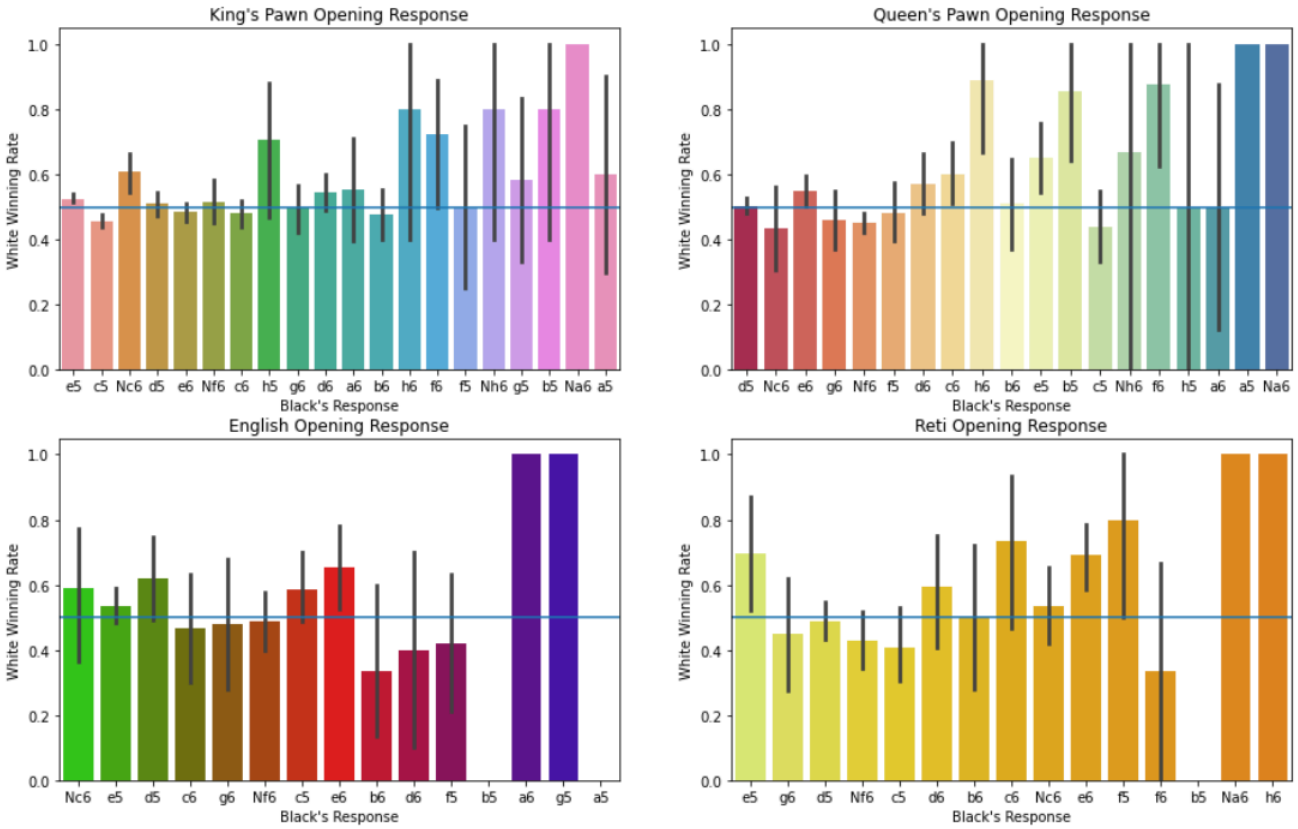


Figure 5.9 Win probability for a opening by black's response

4. Speech Recognition:

- Speech recognition was successful in transcribing user voice commands into accurate chess notations.

- The module effectively distinguished between conformations and rejections, enhancing user engagement.

5. Lichess API Integration:

- The application was very well able to fetch the game data with the help of Lichess API which contained information about the opponents moves and the current game status.

6. Mouse Movements Simulation:

- The application accurately converted the move notations of the starting and end positions of the pieces into board coordinates and with the help of these coordinates mouse movements were simulated onto the screen which enabled the piece to move from one square to another.

Interpretation of Results

The results of the testing indicate that the blind mate chess project is a promising tool for blind chess players. The application is able to accurately identify the chessboard and pieces, recognize move notations and prompts from the player in the form of voice and hand gestures, provide accurate analysis of board positions and puzzles and challenges, and help the player to play a game of chess on an online platform. These findings suggest that the application could be used to help blind chess players learn, practice, and compete in online chess games.

However, there is still room for improvement in the application. The accuracy of the chessboard detection, gesture recognition, piece recognition and speech recognition components could be improved. Additionally, the application could be made more user-friendly and accessible to blind users if there is a possible way to integrate it within the online chess platforms.

5.2 Comparison with Existing Solutions

1. There exists a handful of standalone applications like 'verbal chess' which are not specifically designed for blind people but have the feature of playing online chess with the help of voice commands. The problem with these applications is that they are not integrated with popular online chess platforms like Lichess and Chess.com. They are independent applications which provide the feature of voice command to play chess but only with computers. Most of the online chess users play on Lichess and Chess.com therefore it will be much more useful if the applications can be integrated with these platforms. Our application can be started in the background while playing online chess on Lichess and provide an interface to play on the platform with the help of voice commands.

2. A company with the name of Feelif **[26]** has developed an application Feelif Chess which can be used by people with visual impairments to play chess. This application allows the player to feel the chessboard with chess pieces on the screen and will simultaneously inform the player about the chess piece he/she is touching. They are able to achieve it with the help of a tactile grid which helps the user to feel what is on the screen with the help of vibration and sound. The grid is transparent in nature and consists of dots that are put over the device. A good thing about the application is that since it does not uses any voice command of the player for playing the moves therefore there is no effect of ambient noise on the application, meanwhile our application requires voice commands to play the moves therefore it is sensitive to ambient noise which can hamper the gameplay experience of the blind individual.

3. There are some screen reading applications like NVDA **[27]** and JAWS **[28]** which the blind chess players use to analyze their games. When the blind player touches upon a word on the screen the software informs the blind person about the words on which the cursor has been placed upon. The problem with this approach is that the screen reader is not able to read the position of the piece on the chessboard. An article published by Chessbase India writes about an application developed by Soundarya Kumar Pradhan called Accessible chess that helps to get the position of the pieces with the help of the FEN string of the board position. The entire position is written down and the player can use the screen reader to know about the position. Our application provides the player with the opportunity to get the overview of the chessboard position at any time the person asks for it which makes it very easy for the player to analyze and play the game.

Chapter-06 Conclusion and Future Scope

6.1 Conclusion

The development and testing of the Blind Mate Chess project has culminated in a sturdy and innovative solution designed to empower visually impaired individuals in playing chess on the online Lichess platform. The conclusion summarizes key findings, acknowledges limitations, and highlights the project's contributions to the field.

The following are some of the key findings of the project:

- The chessboard detection component of the application is able to correctly identify the chessboard with an accuracy of about 97%.
- The piece recognition component of the application is able to correctly identify the type of each chess piece with an accuracy of about 95%.
- The speech recognition component of the application is able to correctly transcribe move notations and prompts from the player with an accuracy of about 93%.
- The gesture recognition component of the application is able to correctly identify different gestures with an accuracy of about 92%.

Limitations:

1. **Dependency on Lichess API:**

   The Blind Mate Chess project currently works on only the Lichess platform. It is because the application relies on the Lichess API to fetch the game status and opponent moves. Any kind of Changes or disruptions in the Lichess API may impact the application's functionality. Although Lichess is a reliable platform but it may occur that the API may experience a downtime due to server error which will in turn affect the application’s performance.

2. **No Availability on Mobile Platforms:**

   Currently the application is made by keeping only desktop platforms in focus. The application can not function on android or iOS devices. Online chess platforms also have their app on playstore which can also be used to play chess but currently the application has not been developed to work on mobile platforms.

3. **Sensitivity to Ambient noise**:

The application involves asking chess notations and prompts from the user to ensure that the chess game can be played with the help of voice commands. Although the Whisper AI model used for transcribing speech to text performs very well but is still sensitive to ambient noise. If there is a significant amount of noise in the background the model may not perform accurately and may misidentify the user prompts.

4. Language Dependency:

The speech recognition model may be sensitive to variations in accents, speech patterns, or languages other than English. The model may not be able to correctly transcribe the speech if it is spoken in a different language or with a thick accent. This hampers the inclusivity of the application and makes it limited to players who know certain languages and have a certain accent.

5. Lacks certain necessary features:

The application lacks certain necessary features that are needed so that the blind person can be fully dependent on the application for his gameplay and post game analysis. The application also lacks the feature of providing the player with the chance of resigning or offering a draw in the game. This saves a lot of time for the player. The application also lacks the feature of notifying the player about the current time left on the clock for him to play his move. The player should know the amount of time left on his clock so that he can plan accordingly which move to play in the game. The application is also not able to premove the moves during the game.

6. Challenges in Gesture Recognition:

The application's gesture recognition functionality may face challenges due to various factors such as inconsistent lighting conditions, variations in hand orientation, and limitations in camera quality. Poor lighting conditions can affect the visibility and clarity of hand gestures, leading to inaccuracies in recognition. Similarly, variations in hand orientation, including angles and positions, may impact the model’s ability to correctly interpret gestures. Additionally, the quality of the camera used for capturing hand movements can influence the overall accuracy and reliability of gesture recognition. These factors collectively contribute to potential limitations in the application's ability to accurately detect and interpret user gestures.

Contribution to the field:

The blind mate chess project has made significant contributions to the field of assistive technology for blind individuals by developing an innovative application that enables them to play chess independently. Till now there is no inbuilt tool on online chess platforms that assists people with visual impairments to access the platform's features and make use of them to participate in online chess. The project's contributions are mainly on the sport of chess. The project has addressed the accessibility needs of blind chess players by providing them a tool to engage in online chess independently. This gives them an equal opportunity to participate in online chess tournaments or championships and also to practice and improve their chess skills by giving them the opportunity to play with opponents all around the world. According to an article by Forbes **[29]** in the near future online chess will most probably completely replace on the board chess for major chess events that take place around the world. The project has contributed in providing people with visual impairments an equal chance in competing against players like any other normal individual without any disadvantage. This promotes inclusivity in the sport of chess and empowers blind players to showcase their talent and skills like any other normal person. The blind mate chess projects findings and contributions have laid the foundation for further research and development in the field of assistive technology to aid players in participating and playing online chess.

5.2 Future Scope

1. Cross-Platform Compatibility:

The current application is compatible only with the Lichess platform while there are other platforms also which host online chess tournaments and championships like chess.com. Ensuring compatibility with various online chess platforms will increase the application's accessibility and reach, allowing blind individuals to play chess on a wider range of platforms.

2. Improved Accessibility:

The application can be enhanced to make online chess platforms more accessible to blind players. Currently the application is not able to navigate through various tools of the online chess platform. Natural Language Interaction can be implemented in the application and it can be integrated with the platform's API which will allow the blind player to very effectively navigate the platform and make use of various other tools such as lessons, solving puzzles, and creating challenges provided by the platform.

3. **Expanding Functionality**: The application could be further expanded to include additional features such as:

   - **Game Analysis:** Providing a more in depth post-game analysis to help users learn from their mistakes and improve their chess skills.

   - **Offering or accepting draws and resignations:** Providing players with the opportunity to offer draw or resign during the game as well as accept the opponents draw offers which would save a lot of time in the game.

   - **Notify about time on clock:** Notifying the players about the time left on their clock will help them in planning for future moves in the game and play accordingly the time left on the clock.

   - **Chat and Community Integration:** Integrating a chat feature and community forums within the application, enabling users to interact with fellow players, discuss strategies, and participate in chess-related events.

4. **Exploring Alternative Input Methods for prompt:**

   Currently the prompts of the player are taken with the help of voice commands which can prove to be ineffective sometimes if there is a lot of ambient noise in the background or if the player has a thick and strong accent. If we could develop a hardware device that is suitable for blind players to give their prompts regardless of any ambient noise it would prove to be much more efficient.

5. **Multi-lingual Support and Integration with Online Platforms:**

   Incorporating multilingual support would help blind individuals form various parts of the world with different accents and language to play the game on online chess platforms using their own language as a source of communication. Currently the application is not integrated within any online chess platform. If it is integrated with the platform it will have much more features and will also have access to the platform's internal tools which would make it much more efficient, accurate and easy to use for the player.

References

1. Chess.com, "Chess Is Booming! and Our Servers Are Struggling," Chess.com Blog, Jan. 23, 2023. [Online]. Available: https://www.chess.com/blog/CHESScom/chess-is-booming-and-our-servers-are-struggling

2. J. Balata, Z. Míkovec, and P. Slavík, "Problems of blind chess players," in 2015 6th IEEE International Conference on Cognitive Infocommunications (CogInfoCom), pp. 179-183

3. Lichess, "Lichess API Documentation," June. 20, 2010. [Online]. Available: https://lichess.org/api

4. python-chess, "python-chess: a chess library for Python,"Oct. 26, 2020. [Online]. Available: https://python-chess.readthedocs.io/en/latest/

5. G. Wölflein and O. Arandjelović, "Determining Chess Game State from an Image," Journal of Imaging, vol. 7, no. 6, p. 94, Jun. 2021, doi: 10.3390/jimaging7060094

6. M. Czyzewski, A. Laskowski, and S. Wasik, "Chessboard and Chess Piece Recognition With the Support of Neural Networks," Foundations of Computing and Decision Sciences, vol. 45, pp. 257-280, 2020, doi: 10.2478/fcds-2020-0014'

7. A. Mehta, "Augmented Reality Chess Analyzer (ARChessAnalyzer): In-Device Inference of Physical Chess Game Positions through Board Segmentation and Piece Recognition using Convolutional Neural Network," 2020

8. Chen and K. Wang, "Robust Computer Vision Chess Analysis and Interaction with a Humanoid Robot, "Computers,vol.8,no.1,p.14, Feb. 2019,doi: 10.3390/ computers8010

9. A. de Sá Delgado Neto and R. Mendes Campello, "Chess Position Identification using Pieces Classification Based on Synthetic Images Generation and Deep Neural Network Fine-Tuning," in Proceedings of the 2019 21st Symposium on Virtual and Augmented Reality (SVR),Rio de Janeiro, Brazil, 2019, pp. 152-160,doi: 10.1109 /SVR. 2019.00038.

10. J.Ding, "ChessVision: Chess Board and Piece Recognition, "Stanford Journal of Science, Technology, and Society, 2019"

11. J. Hack and P. Ramakrishnan, "CVChess: Computer Vision Chess Analytics," *Stanford Journal of Science, Technology, and Society*, 2018

References

12. Y. Xie, G. Tang, and W. Hoff, "Chess Piece Recognition Using Oriented Chamfer Matching with a Comparison to CNN," in *Proceedings of the 2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Lake Tahoe, NV, USA, 2018, pp. 2001-2009, doi: 10.1109/WACV.2018.00221

13. Y. A. Wei, T. W. Huang, H. T. Chen, and J. Liu, "Chess Recognition from a Single Depth Image," in *Proceedings of the IEEE International Conference on Multimedia and Expo*, Hong Kong, China, July 10–14, 2017

14. Y. Xie, G. Tang, and W. A. Hoff, "Geometry-based Populated Chessboard Recognition," in *Proceedings of the International Conference on Machine Vision*, 2018

15. C. Koray and E. Sümer, "A Computer Vision System for Chess Game Tracking," in *Proceedings of the IEEE Winter Conference on Applications of Computer Vision*, 2016

16. C. Danner and M. Kafafy, "Visual Chess Recognition," *Stanford Journal of Science, Technology, and Society*, 2015

17. C. Matuszek, B. Mayton, R. Aimi, M. P. Deisenroth, L. Bo, R. Chu, M. Kung, L. Legrand, J. R. Smith, and D. Fox, "Gambit: An Autonomous Chess-Playing Robotic System," in *Proceedings of the 2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 4291–4297

18. N. Banerjee, D. Saha, A. Singh, and G. Sanyal, "A simple autonomous robotic manipulator for playing chess against any opponent in real time," in *Proceedings of the International Conference on Computational Vision and Robotics*, 2011.

19. A. D. Escalera and J. M. Armingol, "Automatic Chessboard Detection for Intrinsic and Extrinsic Camera Parameter Calibration," *Sensors (Basel, Switzerland)*, vol. 10, pp. 2027-2044, 2010

20. R.Kundu,“YOLO: Algorithm for Object Detection Explained”,V7 Labs Blog, Jan.17,2023.[Online].Available:https://www.v7labs.com/blog/yolo-object-detection

21. A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust Speech Recognition via Large-Scale Weak Supervision," 2022

References

22. C. Lugaresi et al., "Mediapipe: A framework for perceiving and processing reality," in *Third Workshop on Computer Vision for AR/VR at IEEE Computer Vision and Pattern Recognition (CVPR)*, 2019, vol. 2019

23. A. Sweigart, "PyAutoGUI," PyPI, 2014. [Online].Available: https://pypi.org/project/PyAutoGUI/

24. Chess.com, "Forsyth-Edwards Notation (FEN)," July 2020. [Online]. Available: https://www.chess.com/terms/fen-chess

25. R.Muthukrishnan and M. Radha, "Edge Detection Techniques for Image Segmentation," *International Journal of Computer Science & Information Technology (IJCSIT)*, vol. 3, no. 6, Dec. 2011, Art. no. 3620-259, doi: 10.5121/ijcsit.2011.3620

26. Feelif Company. (2020) . "Chess" on Feelif. [Online].Available: https://www.feelif.com/tactile-books-games-tools/games/713/chess-8153/

27. NV Access. (2023). *NVDA User Guide*. [Online]. Available: https://www.nvaccess.org/files/nvda/documentation/userGuide.html

28. Freedom Scientific. (2023, March). *JAWS Documentation*. [Online]. Available: https://support.freedomscientific.com/products/blindness/jawsdocumentation

29. M.LoRé, "Online Chess Taking Advantage Of Opportunity To Grow, Entertain During Coronavirus Pandemic," Forbes, May 26, 2020.[Online].Available: https://www.forbes.com/sites/michaellore/2020/05/26/online-chess-taking-advantageofofopportunity-to-growentertainduring-coronavirus-pandemic/?sh=451d8addb974

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: …….

Type of Document (Tick): PhD Thesis | M.Tech/M.Sc. Dissertation | B.Tech./B.Sc./BBA/Other

Name:_____ **Department:**_____ **Enrolment No** _____

Contact No._____ **E-mail.**_____

Name of the Supervisor: _____

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): _____

_____

_____

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

- Total No. of Pages =
- Total No. of Preliminary pages =
- Total No. of pages accommodate bibliography/references =

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at..... (%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor) **Signature of HOD**

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

<table>
<tr><th>Copy Received on</th><th>Excluded</th><th>Similarity Index (%)</th><th colspan="2">Abstract & Chapters Details</th></tr>
<tr><td rowspan="2"></td><td rowspan="4">• All Preliminary Pages
• Bibliography/Images/Quotes
• 14 Words String</td><td rowspan="2"></td><td>Word Counts</td><td></td></tr>
<tr><td rowspan="2">Character Counts</td><td rowspan="2"></td></tr>
<tr><td>Report Generated on</td><td rowspan="2">Submission ID</td></tr>
<tr><td rowspan="2"></td><td>Page counts</td><td></td></tr>
<tr><td></td><td>File Size</td><td></td></tr>
</table>

Checked by
Name & Signature **Librarian**

……

Please send your complete Thesis/Report in (PDF) & DOC (Word File) through your Supervisor/Guide at plagcheck.juit@gmail.com