VOICE-DRIVEN CHESS: ENHANCING GAME ACCESSIBILITY THROUGH SPEECH RECOGNITION AND TEXT-TO-SPEECH INTEGRATION

Ziyi Chai [1], Joshua Lai [2]

[1] Santa Margarita Catholic High School, 22062 Antonio Parkway, Rancho Santa Margarita, CA92688
[2] California State Polytechnic University, Pomona, CA, 91768

ABSTRACT

We aimed to address the lack of accessibility in chess by developing a program that integrates text-to-speech and speech recognition. The system allows users to input moves using voice commands and receive audio feedback of game states, making it helpful for players with visual or motor impairments. The design consists of a central controller to manage game logic, a speech recognizer for move input, and a history manager to track and undo moves. Designing for chess posed numerous edge cases, but we addressed them by building a well-constructed system using proper subclassing and modularity, ensuring flexibility without compromising core functionality. We tested speech recognition extensively across varied inputs, and despite occasional network issues and minor parsing errors, the system achieved very high accuracy and was able to successfully execute most commands the first time, with the remaining commands taking 2-3 more on average. The program is cheap, accessible on most devices, and simple to use.

KEYWORDS

Accessible Gaming, Speech Recognition, Text-to-Speech, Assistive Technology in Chess

1. INTRODUCTION

Chess is a game that is highly dependent on vision. Players rely heavily on the ability to see the board in order to evaluate positions, anticipate moves, and strategize effectively. Visual cues play a central role in recognizing patterns, calculating variations, and managing time. However, many popular online chess platforms lack essential accessibility features, such as voice feedback or voice recognition controls [1]. For example, as of June 2025, Chess.com—the world's most popular online chess platform with over 200 million users—does not offer built-in text-to-speech or voice recognition functionalities to support users with visual impairments [2][3].This becomes even more challenging when compared to physical chess sets, where players with visual impairments can use tactile boards and pieces to feel their way through a game [4]. On a website, that tactile interaction is lost, creating a significant barrier. For the estimated 2.2 billion people worldwide who experience some form of vision impairment or blindness, these design oversights can make virtual chess inaccessible or extremely difficult to play. This not only limits their ability to participate in the game recreationally but also excludes them from competitive or educational opportunities that chess offers. By integrating inclusive features such as audio descriptions, keyboard navigation, voice commands, and compatibility with assistive

David C. Wyld et al. (Eds): CMLA, CSEIT, NLPD, GRAPH-HOC, SPM, NeTCoM, NCS, ArIT, DMSE – 2025
pp. 53-63, 2025. CS & IT - CSCP 2025 DOI: 10.5121/csit.2025.151305

technologies, chess platforms can become more accessible [5]. Making these changes would promote inclusivity, ensuring that chess can truly be enjoyed by everyone, regardless of visual ability.

Other researchers have also attempted to address the same issue. In 2005, Grammenos created a chess application that allowed users to play the game with some simple accessibility features. While this was well built for the time, it is outdated as it cannot operate on modern operating systems. In contrast, our projectworks on modern Windows systems and can also be built for macOS or other modern operating systems. Another effort came from Anandaraj et. al., who created a chess clock with text-to-speech capabilities, aiming to assist visually impaired individuals. However, this tool is limited to announcing the turn number and remaining time;it cannot interpret or describe the game state itself. Our system addresses these limitations by announcing the specific moves and also allowing users to utilize voice commands. Finally, Guerini created a robot to move physical pieces for disabled individuals. While this technology is certainly impressive, the robot itself is not affordable, costing around 50,000 euros. Our systemis a piece of open-source software, allowingeasy distribution while maintaining compatibility.

Overall, Chess4All aims to expand accessibility in chess, particularly for visually impaired players. It is a piece of software that allows people to play chess without a physical board. Traditional chess websites often lack inclusive features that allow people with vision impairments to enjoy the game. To address this issue, we implemented Text-to-Speech functionalities, which enable users to understand what moves are happening in the game without needing to see the board [6]. In addition, it also features speech recognition, allowing players to speak any move they wish to make, whether that be moving a piece, promoting, or undoing their most recent move. Of course, it also has more traditional controls for sighted players as well, and can handle all standard chess edge-cases (castling, En-Passant, etc.). It also features FEN compatibility, meaning that board states can be exported to or imported from any other chess software using the FEN standard. Other solutions currently available are either old and outdated, require expensive hardware, lack key accessibility features, or are otherwise inconvenient to use. Our project, on the other hand, is easily accessed, simple to use, and requires no extra hardware besides a standard Windows, Linux, or Mac device (which most people have nowadays) [7].

In our experiment, we aimed to test the speech recognition capabilities of our program. We wanted to know whether or not it would hold up in regular use, and if there were any moves or commands in particular it struggles with. In order to set this up, we prepared a list of various commands, splitting them into three categories: Movement (moving and taking pieces), Promotion (promoting and cancelling promotions), and Miscellaneous (undoing moves and filtering out gibberish). We then tested each command, recording whether it succeeded on the first try and how many times it took to succeed. We discovered that around 83% of attempts were executed successfully on the first try, and of the ones that failed, they only needed an average of 2-3 more tries to succeed as well. This experiment was very successful, illustrating that our system has relatively high accuracy with a variety of normal spoken inputs.

This paper is structured as follows. Section 2 presents key challenges and critiques encountered in designing the system. Section 3 provides an overview of the solution architecture and the structure of major components. Section 4 describes the experimental setup and results. Section 5 reviews related work and how our solution compares. Section 6 concludes the paper and outlines future work.

2. CHALLENGES

In order to better understand how users interact with the program, a few frequently raised challenges and questions have been identified.

2.1. Board Controller

Challenge:

"How effectively does the engine handle edge cases, like en passant moves?"

Response:

Edge cases involving en passantare accounted for. It uses a "ghost piece" to internally represent this space that can be captured behind the pawn, and this piece is registered on the board as the same as the piece ahead of it. If it is captured, the corresponding piece is captured as well. Ghost pieces are also created if they are defined in an FEN string, so the entire game state can be recorded and imported. Ghost pieces are then removed at the end of each turn to ensure they cannot be exploited in unexpected ways. Other non-standard moves, such as castling, are also handled.

2.2. Speech Recognition Module

Challenge:

"Aren't speech recognition models computationally expensive? How can you ensure that all people can run the game? Also, how well does it work?"

Response:

Instead of processing the spoken audio on the user's device, we instead use an API (Application Programming Interface) and offload the computational task to a remote server with much higher processing power than most users' devices. This way, they can still input commands at a reasonable rate, without their device overheating or slowing down. As for the second question, our system is fairly accurate, as illustrated by the experiment detailed above. 83% of tested commands executed successfully on the first try, and even for the remaining 17%, repeating the command with a clear and level tone 2-3 more times resulted in success.

2.3. History Manager

Challenge:

"Wouldn't it take a lot of data to store the entire chess history on the user's device? Also, why is it important to use FEN?"
Response:

Storing chess history requires minimal data. Each move in the game history requires less than a kilobyte in memory, meaning that an entire chess game can be stored in just a few kilobytes. For perspective, the image below is approximately 62 kilobytes, which is easily downloaded and stored by any modern computer.

FEN is the standard for storing an entire chess game, as it contains all information about a game state. This includes the position of all of the pieces on the board and any miscellaneous information such as the availability of a castling move, any possible enpassant, and the active player. Our system scans through the board and records every detail into an FEN string, meaning it can be exported to any other chess software that supports the FEN standard.

3. SOLUTIONS

The program initializes and displays a chessboard as the first screen. The user can choose to start a game or paste in an FEN (Forsyth-Edwards Notation) that denotes a chess position [8]. The game would recreate the given position. There is also a display for a PGN, which tracks all moves made in the chess game currently. A Text-to-Speech program announces the move that was just played. Should the player wish to mute the TTS voice, there is a checkbox in the settings menu that allows them to do so [9]. To play the game, players click on chess pieces, which highlight where the piece may move (indicated by small grey dots) and what pieces it may capture (indicated by a highlighted red square). After the player makes their selection, the piece is moved, and the opposing player now takes their turn. If a pawn reaches the opposite side of the chessboard, an overlay will appear over the screen that contains four options for the player to promote their pawn to (Queen, Knight, Rook, Bishop). At the top right of the screen is an undo button, which undoes the most recent move. The game keeps a history of all the moves that have been performed, so this undo function can be used repeatedly. If at any point in the game, a checkmate or statement is reached, the game will end, and a screen will pop up to notify the players that the game is over. Finally, in addition to mouse controls, speech commands can also be used to play the game.

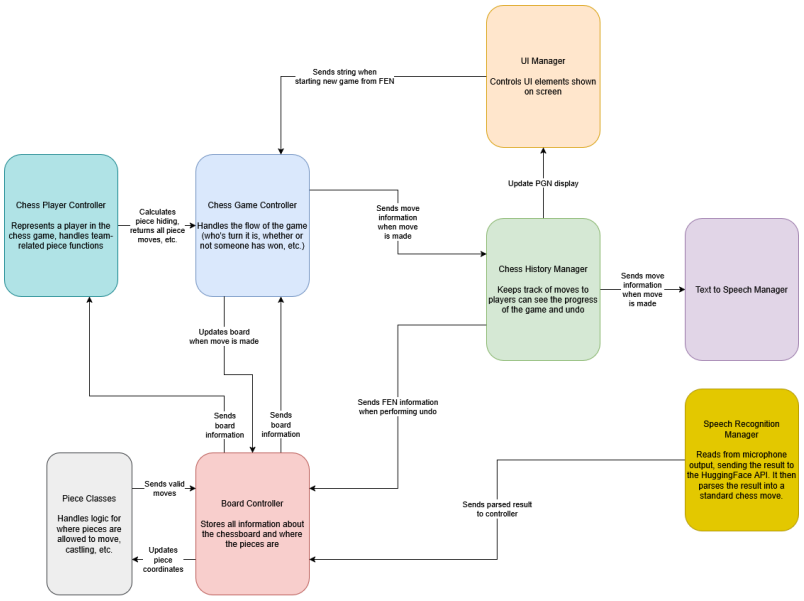

Figure 1. Overview of the solution

The main component is the Chess Game Controller. This is the largest and most complex component in the system by far, and it communicates closely with the board controller and the history manager. It handles everything related to the overall chess game functionality, including creating the pieces when a game is initialized, managing which player's turn it is, or checking all the conditions to see if a game is finished.

Figure 2. Screenshot of the chess

Code removed for brevity—
See ChessGameController.cs on GitHub.

Figure 3. Screenshot of code 1

The code above is an excerpt from the two functions used to check if the game has ended. This is called right after a player has made a move (for simplicity, the player who just moved will be referred to as the attacker, and the player who will move next will be referred to as the defender). First, the manager calls the defender to both remove any of their own moves that could endanger

their own king. Next, it generates all possible moves for both players and finds if the defender's king is threatened by any of the attacker's pieces. If so, it calculates if there are any moves the defender could do to save their king (blocking it with another piece, for example). If not, it returns a checkmate and terminates the game. If there's nothing the player can do to save their king, but there's also no pieces currently threatening it, a statement is called instead. Otherwise, it returns that the game is still running and swaps control to the next player.

The Speech Recognition Module handles speech recognition, allowing users to input moves without needing to physically interact with the device. It uses HuggingFace's API to process audio clips into readable strings of text. Using this system, users can make moves, undo moves, handle promotions, or cancel actions.

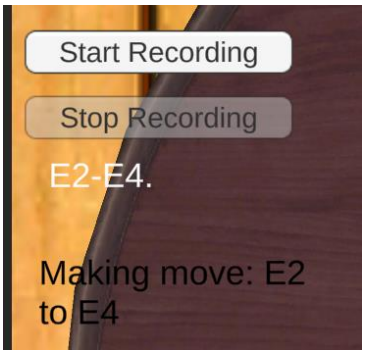

Figure 4. Screenshot of recording

Code removed for brevity—
See SpeechRecognitionModule.cs
on GitHub.

Figure 5. Screenshot of code

The system first begins recording audio from the microphone, storing it in a buffer. Once the recording has completed, it is encoded into a .WAV byte array and sent to the HuggingFace API [10]. The API then returns a text string. The function above parses through this text string and reads the commands it contains. First, it uses a regular expression to search for moves in the format [LETTER] [NUMBER] … [LETTER] [NUMBER]. If it finds these moves, it converts them to a compact string and sends it to the input handler. Otherwise, it searches for a piece name, such as "Queen" or "Knight." If it can find one of such, it executes a promotion check, resulting in a promotion if valid. If it cannot find those keywords, it tries searching for the phrases "Undo" and "Cancel", and calls their respective commands if they are found. Otherwise, it outputs a simple message saying that the command could not be parsed.

The Chess History Manager keeps track of every move that was played in a game, as well as acting as a translation between PGN and FEN. It contains an internal list of moves, recording information such as the piece that moved, where it moved, pieces it captured, whether it caused a checkmate, the FEN string associated with the board position, and more.

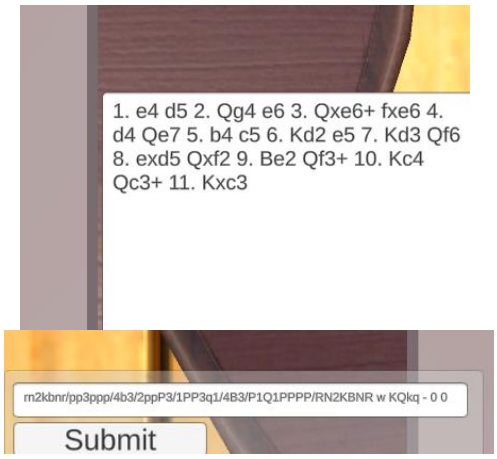

Figure 6. Screenshot of the PGN

Code removed for brevity—
SeeChessHistoryManager.cs on GitHub.

Figure 7. Screenshot of code 3

The Chess History Manager is responsible for tracking and displaying past moves, which allows players to view, undo, and restore previous positions. It uses the GetFEN method to maintain a history of moves in Forsyth-Edwards Notation (FEN), a method to record the information and state of a game at a given move. The function begins by iterating over the entire board, reading in every piece as a corresponding letter in the string. As it does so, it also notes any available spaces for en passant. Once it finishes scanning the board, a string is constructed, consisting of the piece positions followed by the castling rights, en passant squares, and move counters. By storing each move in a listusing FEN format, the Chess History Manager ensures that previous positions can be recreated, allowing users to undo moves or export the position so it can be viewed elsewhere.

Code availability

All source code for this project is open-source and publicly available on GitHub at: https://github.com/sbxdmz/Chess-Board. All users and developers are encouraged to explore and contribute to the code to improve accessibility in chess software.

4. EXPERIMENT

The purpose of this test is to determine how a user reacts to the program. This is important, as the creators of a program can often miss UX issues, since they have become accustomed to how the program operates. Testing out a project with users who are unfamiliar with the software helps identify points that are confusing, frustrating, or otherwise impact the user experience.

We used a variety of inputs, trying a mix of numerical, letter, and word commands. We needed to ensure that the system worked in as many situations as possible, as users tend to be inconsistent and will say more than just the commands. We tested both the system's ability to recognize keywords, as well as its ability to respond appropriately given the game state. There may be cases where a command is recognizable, but cannot be performed (i.e., attempting to promote a piece when there is no pawn on the opposite side). The system should be able to detect this and thus avoid executing illegal commands.

Experiment	Category	Command	Expected	Times needed to successfully execute command	First try?
1	Moving Pieces	“E2 to E4”	Moves piece on E2 to E4	1	TRUE
2	Moving Pieces	“E2 to uhhhhhhh let me think… uh… E4…. yeah.”	Moves piece on E2 to E4	1	TRUE
3	Moving Pieces	“E2 to E4”, except there is no piece on E2	Not do anything	1	TRUE
4	Moving Pieces	“E2 to E4”, except one of your pieces is on E4	Not do anything	1	TRUE
5	Moving Pieces	“E1”	Not do anything	1	TRUE
6	Moving Pieces	“E2E4”	Moves piece on E2 to E4	1	TRUE
7	Moving Pieces	“Knight B1 to C3”	Moves piece on B1 to C3	1	TRUE
8	Moving Pieces	“H2 to H3”	Moves piece on H2 to H3	1	TRUE
9	Moving Pieces	“H7 to H8”	Moves piece on H8 to H7	3	FALSE
10	Moving Pieces	“I want to move the piece on B1 to B4.”	Move piece on B1 to B4	2	FALSE
11	Promotion	“Promote to Queen”	Promotes the piece to a queen	1	TRUE
12	Promotion	“Promote to Pawn” when a promotion move is valid	Not do anything	1	TRUE
13	Promotion	“Promote the piece to Knight”	Promote to a knight	4	FALSE
14	Promotion	"Cancel" when not promoting	Do nothing	1	TRUE
15	Promotion	“Cancel” when promoting	Cancel promotion	1	TRUE
16	Promotion	"I want to promote to Queen"	Promote to queen	1	TRUE
17	Promotion	"Promote my pawn into a Queen."	Promote to queen	1	TRUE
18	Promotion	"Please perform a promotion to rook"	Promote to rook	3	FALSE
19	Promotion	"Promote to bishop"	Promote to bishop	1	TRUE
20	Promotion	"Promote."	Do nothing	1	TRUE
21	Misc	“Undo” when a move has been made	Undo move	1	TRUE
22	Misc	"I want to undo."	Undo move	1	TRUE
23	Misc	“Undo” when no moves have been made	Do nothing	1	TRUE
24	Misc	“Hello world!”	Do nothing	1	TRUE
25	Misc	"blah blah blah blah blah"	Do nothing	1	TRUE
26	Misc	"Oops I didn't mean to do that"	Do nothing	1	TRUE
27	Misc	"Let me undo the move I just made."	Undo move	2	FALSE
28	Misc	"Undo please."	Undo move	1	TRUE
29	Misc	"Undo Undo"	Undo move	1	TRUE
30	Misc	"I love chess!"	Do nothing	1	TRUE

Figure 8. Table of Experiment 1

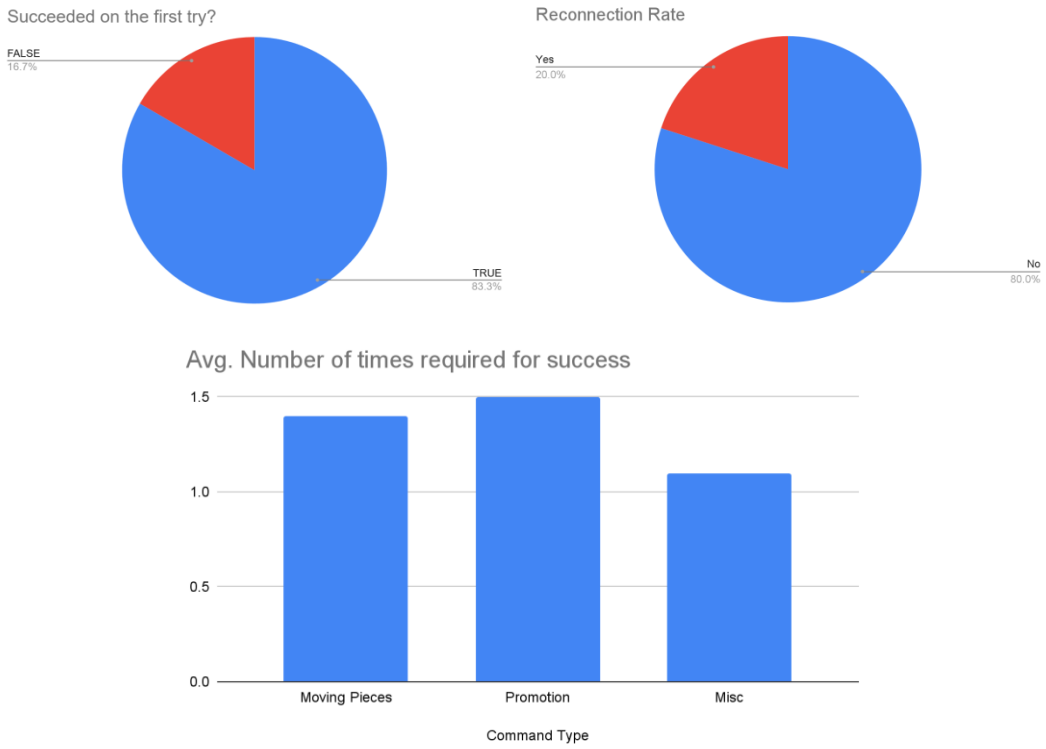

Figure 9. Figure of experiment 1

Out of the 30 inputs we provided, our system correctly identified and executed 24 of them on the first attempt, achieving an accuracy of 83%. It is important to note that the inputs that failed the first time were able to be executed after 1-3 more tries while speaking slowly and clearly. One common factor between these failed inputs is that they are mostly long and verbose. This seems to cause the system to attempt to analyze the command as a sentence instead, resulting in it reading numbers as words (i.e. “7” being read as “SEVEN”) or reinterpret certain words based on what would be more common for a sentence (i.e. “To Knight” read as “Tonight”).

Promotion appeared to have the most failures, which can be attributed to the fact that it needed to identify words that are not as commonly used (like Rook and Bishop). The miscellaneous commands had the highest rate of success and required the least number of repeated commands, likely because all the system must do for many of these tests is recognize that there is no valid command within the sentence. Around 20% of attempts had some minor connection errors, due to the API taking a few seconds to refresh before the new request.

5. RELATED WORK

This paper was written by Dimitris Grammenos in 2005 [11]. It is intended to solve a problem similar to the one we are trying to solve, which is to allow people with disabilities to play the game. In order to achieve this, they created a piece of software that simulates a chessboard, allowing players to move pieces using the mouse, keyboard, or speech recognition. One major advantage that our project has over this one is a simple matter of hardware compatibility. This project, though well-made for the time, is undeniably outdated and does not run on modern systems. Ours, on the other hand, is able to operate on all modern Windows systems.

This paper by Praveen Raaj Anandaraj, Nasrul Humaimi Mahmood, Mohd Azhar Abdul Razak, and Nor Aini Zakaria aims to assist visually impaired chess players by replacing the traditional chess clock used in competitions with one that provides audio callouts as the game progresses [12]. This device is very limited in its capabilities, as it is only able to provide information such as time remaining or the number of moves that have occurred. Furthermore, this device alone does not allow the user to play chess. While it aids the game, a physical board is still needed. Our project, on the other hand, not only provides an environment to play the game, but also gives much better detail in the text-to-speech portion, as it can describe moves as they happen (Queen to E5, for example).

This paper by Silvia Guerini describes a chess robot used to allow individuals with severe motor impairment to play chess [13]. The robot can be controlled via a mouse, touchpad, or joystick, and is able to move chess pieces across a board. One major drawback of this system is its affordability. The robot used for this paper is the PAL Robotics TIAGo, which costs around 50,000 euros– even without the chess software. This is obviously outside the budgets of most individuals. Our project, on the other hand, runs on any standard PC device, which most people have access to. No new equipment or apparatuses are required for the project, making it extremely accessible.

6. CONCLUSIONS

If given more time, we would focus on expanding the program to include an online multiplayer. One such feature could be a competitive option, allowing users to play against one another. This would include different time controls and game paces that users can select. Additionally, making user games open-source would aid in developing training courses and chess puzzles. Users would also be able to replay and analyze their previous games through a database.

Another feature we aim to develop is a physical board that is compatible with the program. This board would allow for even more accessibility features. Pieces and the board could include tactile indicators or automatically moving pieces, both of which would significantly improve the playing experience for visually impaired users.

One limitation of our project was that the speech recognition system relies on an API. Though it avoids heavy computations on the user's device, it also requires an active internet connection for the system to function. If we had more time, we could create our speech recognition model, which could be downloaded and used should the API fail.

Another limitation is that the text-to-speech system uses Windows TTS, so this is the only part of our program that is not compatible with other systems. To remedy this, we could use an API to generate voice clips, similar to the way the speech recognition system operates currently, or we could rely on an offline TTS package that can be bundled with the game itself.

Overall, Chess4All demonstrates the impact that accessibility can bring to the game of chess. With further development, it has the potential to transform the way visually impaired users interact with the game.

REFERENCES

[1] Ali, Awadalla Taifour, E. B. Eltayeb, and Esra Altigani Ahmed Abusail. "Voice recognition based smart home control system." International Journal of Engineering Inventions 6.4 (2017): 01-05.

[2] Waldstein, David (March 15, 2020). "Think Cheating in Baseball Is Bad? Try Chess". *The New York Times*. ISSN 0362-4331. Retrieved December 22, 2021.

[3] Chess.com Help & Support Center. Accessed June 29, 2025. https://support.chess.com
[4] Mikhaylova, I. V., A. S. Makhov, and A. I. Alifirov. "Chess as multicomponent type of adaptive physical culture." Theory and practice of physical culture 12 (2015): 56.
[5] Mallasén, David, et al. "Exploring Low-Cost Platforms for Automatic Chess Digitization." Journal of Computer Science & Technology 25 (2025).
[6] Nagdewani, Shivangi, and Ashika Jain. "A review on methods for speech-to-text and text-to-speech conversion." International Research Journal of Engineering and Technology (IRJET) 7.05 (2020).
[7] Adekotujo, Akinlolu, et al. "A comparative study of operating systems: Case of windows, unix, linux, mac, android and ios." International Journal of Computer Applications 176.39 (2020): 16-23.
[8] Iqbal, Azlan. "An algorithm for automatically updating a Forsyth-Edwards notation string without an array board representation." 2020 8th International Conference on Information Technology and Multimedia (ICIMU). IEEE, 2020.
[9] Cambre, Julia, et al. "Choice of voices: A large-scale evaluation of text-to-speech voice quality for long-form content." Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems. 2020.
[10] Jain, Shashank Mohan. "Hugging face." Introduction to transformers for NLP: With the hugging face library and models to solve problems. Berkeley, CA: Apress, 2022. 51-67.
[11] Grammenos, Dimitris, Anthony Savidis, and Constantine Stephanidis. "UA-Chess: A universally accessible board game." Proceedings of the 3rd International Conference on Universal Access in Human-Computer Interaction, Las Vegas, Nevada (July 2005). 2005.
[12] Anandaraj, Praveen Raaj, et al. "Digital Chess Clock for Visually Impaired Players." Journal of Human Centered Technology 3.1 (2024): 46-52.
[13] Chen, Andrew Tzer-Yeu, and Kevin I-Kai Wang. "Robust computer vision chess analysis and interaction with a humanoid robot." Computers 8.1 (2019): 14.