AN ADAPTIVE CHESS-BASED LEARNING PLATFORM FOR CHILDREN WITH ACCESSIBILITY FEATURES USING GENERATIVE AI AND MACHINE LEARNING

Qixin Lin [1], Owen Miller [2]

[1] Central High School, 1700 W Olney Ave, Philadelphia, PA 19141
[2] Computer Science Department, California State Polytechnic University, Pomona, CA 91768

ABSTRACT

Chess is a widely played game, but accessibility barriers can prevent many people from fully engaging with it. This project aims to study and create a more accessible chess application that integrates AI features to help with usability. The program features 3 main systems, a speech-to-command system that allows players to make moves using voice input, a text-to-speech system to give audio feedback on board states, and a system for move validation to make sure the AI output is more accurate [1].

To test the system, we did an experiment where users provided voice inputs, and we compared them to what the AI thought they said to look for the most common mistakes in the AI's output. This showed that most simple commands were accurately processed, but more complex phrases and words that could be spelled in multiple ways gave it more trouble [9]. We were able to use those tests to look for cases of slight errors and adjust for them to improve our speech command's accuracy. This project helps demonstrate how AI features can enhance accessibility features to promote more inclusive gameplay.

KEYWORDS

Accessible Chess, AI-Powered Usability, Speech-to-Command System, Inclusive Gameplay

1. INTRODUCTION

I was inspired to create this chess app because I want people all around the world to experience and play chess as I've met people with sensory impairment(s) before who still love and play chess [2]. However, because people who unfortunately have sensory impairment(s) have a hard time learning, and improving at chess. It is estimated that 5-16.5% of the general population has a sensory impairment, an extreme detriment to their lives as the loved ones of people with serious sensory impairments must take care of them, negatively affecting the lives of their own, and their loved ones [3]. My hope with this chess app is to be one of the positive things that people with sensory impairments have.

The articles talk about ways they have explored and implemented different accessibility features for both physical and digital games. The annotated chess board solves the issue of knowing different locations of a chess board by feeling it, but this does not transfer to digital games since most games do not have physical objects to support interactions [10]. The issue the maze game ran into is the delay with their audio commands and the actual input going through. This is the biggest downside with speech to commands, whatever the spoken command is has to be fully spoken before an action can start to take place. This is not as present with our because the game does not have a time limit and allows the player to play at the pace they choose.

The project idea is to adapt an existing chess project to introduce more accessibility features into the game. We are using an existing game of chess as a base instead of creating one from scratch because we want to focus more on creating features that can enhance the game instead of creating the game ourselves. We created a speech-to-commands feature so that instead of having the "grab" pieces and move them to the position you want, the user can instead state "A2 to A3" and the piece on the a2 square will be moved to the a3 square. This feature is not limited to just moving pieces, menus and other settings can be changed using it. This allows users to interact with the game with minimal button presses. There is also a system to have those moves read back to the user once the move is completed. This solution is much more effective in the context of chess. This is because using speech commands takes time to execute, in the context of chess, users will have more time between moves to both think of their move and give the system time to process their move instead of using speech to commands in a faster-paced game.

We tested the AI system's accuracy. We gave the system a set of voice data to see how it handled the input and the accuracy of its output. We found that it was mostly accurate at interpreting the speech into words, but it would often get confused and return words that were not said. This made us change how we use the AI's output by filtering its output and correcting it. We looked for common mistakes and made cases to correct those mistakes. The experiment helped us refine the audio system even more so that we can make more accurate decisions on the system's output.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. The Speech-to-Commands Feature

For the speech-to-commands feature, skepticism could be that it would struggle to understand words that some people say and would not be able to parse their speech into a command or move. We have addressed this by adding an accepted word list. We noticed it was struggling with some words when navigating menus so when a menu command is said it goes through the list and sees if the word it heard is on the list, such as "text to speech" could be heard as "texas beach," we tested the phrases and added similar sounding ones to the list.

2.2. The Text-to-Speech Feature

The second feature is the text-to-speech. One concern is that the voices can sound unnatural or robotic and negatively impact user experience. One of the solutions is we can allow users to change the speaking voice to one of the other characters to make the voices more unique.

2.3. Convert the Output

The last feature is the way that we convert the output of the AI to make sure the moves come out accurately. We solve this concern by having the commands structured in a predictable format, "A2 to A3." Nowe that the commands are in a predictable format, we can account for situations where the output is just a little bit inaccurate.

3. SOLUTION

First, the user has the option to start using the speech-to-commands, to use this the user has to press space bar once to start speaking a command, and once more to finish the command. The audio is sent to HuggingFace to turn the audio into text which will then come back [4]. The text is then parsed, and the code completes a command based on what the text came back to. There is also a word list that can be found in the upper right corner to see what types of words can be used to navigate menus. There are 2 play options, the first is the Sandbox mode, which gives all the pieces the ability to move whenever they want, this can also be used to play locally with people [5]. The other mode is the play-against-AI mode. The user can choose if they want to play as white or black pieces for who goes first, the user can also choose the difficulty of the AI opponent from 1 to 3. When inside the game, based on piece color one person will go first and start the game. The last button on the home menu is the quit button, it will shut down the entire game.

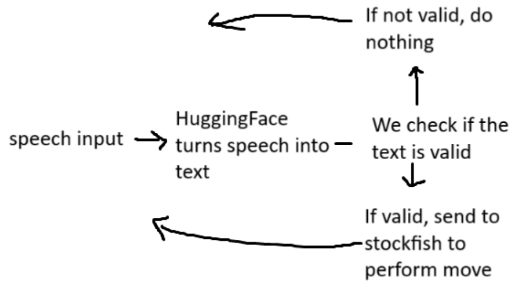

If not valid, do nothing
speech input
HuggingFace turns speech into text
We check if the text is valid
If valid, send to stockfish to perform move

Figure 1. Overview of the solution

The first component's purpose in my program is voice command [6]. Voice command listens to user input by using an API that turns our voice into a string.

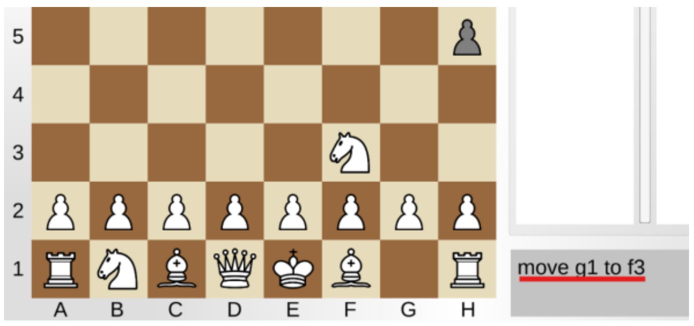
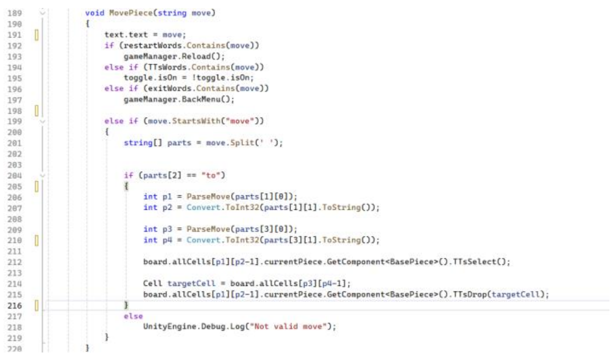


Figure 2. Screenshot of the game

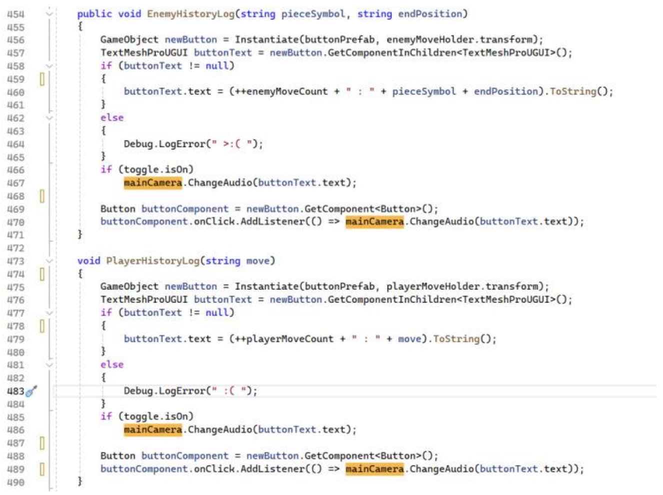


```
void MovePiece(string move)
{
    text.text = move;
    if (restartWords.Contains(move))
        gameManager.Reload();
    else if (TTsWords.Contains(move))
        toggle.isOn = !toggle.isOn;
    else if (exitWords.Contains(move))
        gameManager.BackMenu();

    else if (move.StartsWith("move"))
    {
        string[] parts = move.Split(' ');


        if (parts[2] == "to")
        {
            int p1 = ParseMove(parts[1][0]);
            int p2 = Convert.ToInt32(parts[1][1].ToString());

            int p3 = ParseMove(parts[3][0]);
            int p4 = Convert.ToInt32(parts[3][1].ToString());

            board.allCells[p1][p2-1].currentPiece.GetComponent<BasePiece>().TTsSelect();

            Cell targetCell = board.allCells[p3][p4-1];
            board.allCells[p1][p2-1].currentPiece.GetComponent<BasePiece>().TTsDrop(targetCell);
        }
        else
            UnityEngine.Debug.Log("Not valid move");
    }
}
```

Figure 3. Screenshot of code 1

The function receives a command, the command is a string that is assigned to the variable move. First, the text is displayed in the game so that the user can see what the AI heard them say, then it compares the text to word lists to find out if the user is trying to navigate a menu. If the user is not trying to navigate a menu and is trying to move a piece then it goes to the next section of the code. If the first word is “move” then the code knows that it's a move, if the first word is not “move” then the code exits. If it is a move then the words are split up into sections, the first section is for the starting position and the second section is for the end position of the piece, and it attempts to do the move. If nothing happens then the move is invalid and the user can try another move.

The second component is a text-to-speech function [7]. The TTS will say the move that has just been played as it is happening and it can be prompted to repeat the move if the user clicks on the move in either of the move history panels.



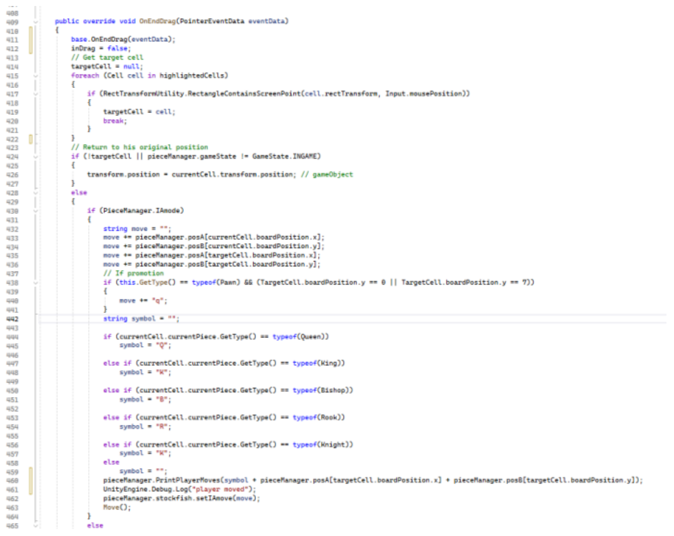
```
public void EnemyHistoryLog(string pieceSymbol, string endPosition)
{
    GameObject newButton = Instantiate(buttonPrefab, enemyMoveHolder.transform);
    TextMeshProUGUI buttonText = newButton.GetComponentInChildren<TextMeshProUGUI>();
    if (buttonText != null)
    {
        buttonText.text = (++enemyMoveCount + " : " + pieceSymbol + endPosition).ToString();
    }
    else
    {
        Debug.LogError(" >:( ");
    }
    if (toggle.isOn)
        mainCamera.ChangeAudio(buttonText.text);

    Button buttonComponent = newButton.GetComponent<Button>();
    buttonComponent.onClick.AddListener(() => mainCamera.ChangeAudio(buttonText.text));
}

void PlayerHistoryLog(string move)
{
    GameObject newButton = Instantiate(buttonPrefab, playerMoveHolder.transform);
    TextMeshProUGUI buttonText = newButton.GetComponentInChildren<TextMeshProUGUI>();
    if (buttonText != null)
    {
        buttonText.text = (++playerMoveCount + " : " + move).ToString();
    }
    else
    {
        Debug.LogError(" :( ");
    }
    if (toggle.isOn)
        mainCamera.ChangeAudio(buttonText.text);

    Button buttonComponent = newButton.GetComponent<Button>();
    buttonComponent.onClick.AddListener(() => mainCamera.ChangeAudio(buttonText.text));
}
```

Figure 4. Screenshot of code 2

These are both functions that control the move history tabs for both the player and opponent. The format of a move that goes into the history panels is the number that the move is, if it's the first move it's 1, if it's the second move it's 2, then the piece symbol and its end position. For example 2 : Kh3 | means it's the second move and the knight was moved to k3. When it receives that data it creates a button that displays that information, the reason the data is on a button is so that it can be clicked and the text will be repeated. Once the button is created it then calls a function called

ChangeAudio that is attached to the camera. When that function is called the audio of the movie that was just made is played from the camera so the player can hear it.

The third component is the component that is responsible for moving the pieces. Every time a piece is clicked on the OnBeginDrag function is called, when the piece is dropped the function OnEndDrag is called to place the piece, this is the OnEndDrag function.



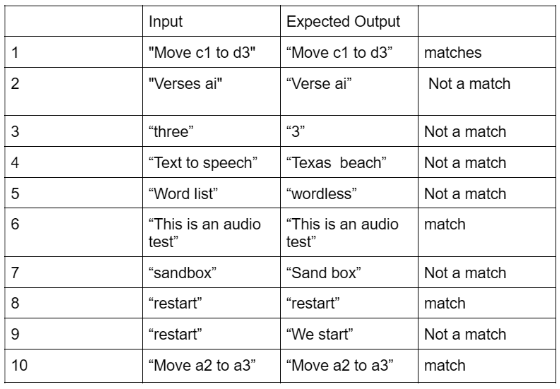
```
public override void OnEndDrag(PointerEventData eventData)
{
    base.OnEndDrag(eventData);
    inDrag = false;
    // Get target cell
    targetCell = null;
    foreach (Cell cell in highlightedCells)
    {
        if (RectTransformUtility.RectangleContainsScreenPoint(cell.rectTransform, Input.mousePosition))
        {
            targetCell = cell;
            break;
        }
    }
    // Return to his original position
    if (!targetCell || pieceManager.gameState != GameState.INGAME)
    {
        transform.position = currentCell.transform.position; // gameObject
    }
    else
    {
        if (PieceManager.IAmode)
        {
            string move = "";
            move += pieceManager.posA[currentCell.boardPosition.x];
            move += pieceManager.posB[currentCell.boardPosition.y];
            move += pieceManager.posA[targetCell.boardPosition.x];
            move += pieceManager.posB[targetCell.boardPosition.y];
            // If promotion
            if (this.GetType() == typeof(Pawn) && (TargetCell.boardPosition.y == 0 || TargetCell.boardPosition.y == 7))
            {
                move += "q";
            }
            string symbol = "";

            if (currentCell.currentPiece.GetType() == typeof(Queen))
                symbol = "Q";

            else if (currentCell.currentPiece.GetType() == typeof(King))
                symbol = "K";

            else if (currentCell.currentPiece.GetType() == typeof(Bishop))
                symbol = "B";

            else if (currentCell.currentPiece.GetType() == typeof(Rook))
                symbol = "R";

            else if (currentCell.currentPiece.GetType() == typeof(Knight))
                symbol = "N";
            else
                symbol = "";
            pieceManager.PrintPlayerMoves(symbol + pieceManager.posA[targetCell.boardPosition.x] + pieceManager.posB[targetCell.boardPosition.y]);
            UnityEngine.Debug.Log("player moved");
            pieceManager.stockfish.setIAmove(move);
            Move();
        }
        else
```

Figure 5. Screenshot of code 3

When a piece is picked up it colors all of the squares the piece is allowed to be placed on. The first thing the OnEndDrag function does is loop through all of the highlighted squares and make sure that the player is dropping the piece on one of the highlighted squares, if the square is not highlighted nothing happens. If it is highlighted then it starts to try and build out a move history entry by checking what type of piece the currently selected piece is. After it figures it out, the move is sent to the pieceManager to create a move history entry for it. It then calls the stockfish API to perform the move, stockfish checks if there is an opposing piece in the slot and if there is the piece is captured, if there isn't then the piece is moved to the square.

4. EXPERIMENT

The project uses voice recognition for speech to commands, for navigating menus and for controlling chess pieces. We are going to test the accuracy of the Ai by comparing the output text compared to what the user actually said [8].



	Input	Expected Output	
1	"Move c1 to d3"	“Move c1 to d3”	matches
2	"Verses ai"	“Verse ai”	Not a match
3	“three”	“3”	Not a match
4	“Text to speech”	“Texas beach”	Not a match
5	“Word list”	“wordless”	Not a match
6	“This is an audio test”	“This is an audio test”	match
7	“sandbox”	“Sand box”	Not a match
8	“restart”	“restart”	match
9	“restart”	“We start”	Not a match
10	“Move a2 to a3”	“Move a2 to a3”	match

Figure 6. Table of experiment

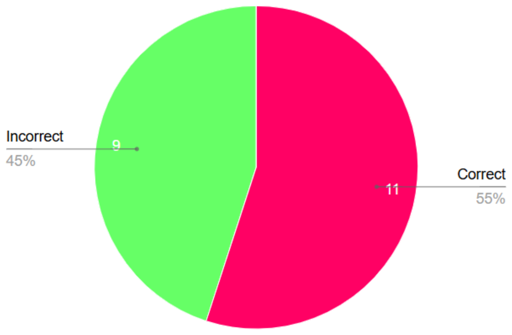


Figure 7. Table of the experiment

Even though most of the cases shown above were of not matches, the system did a good job of matching its output with what was originally said. We wanted to highlight that at times the system's output was incorrect and led to having to say the same commands multiple times. The best way to make sure an output was accurate was by speaking slowly and clearly which made it feel slightly unnatural to use at times. Even though there are not that many commands in the project that we would use, the system when talking normally was about 60% accurate, but when not thinking about the way I speak dropped to around 40%. While the system is useful, it would not be entirely practical to use in every case because of its inaccuracies.

5. Related Work

This is a link to the home page of the International Braille Chess Association, their goal is to help make chess more accessible to blind people [11]. This is similar to the one of this project, they want to make chess more accessible and bring it to a larger audience, this project works with digital chess, but theirs is more focused on assisting with a physical board. They make the board more readable by annotating it in different ways with different markings, we use voice commands and text-to-speech so that the user does not have to rely as heavily on the screen.

This article, unlike the other, focuses more on accessibility for digital games [12]. Like us, they are using voice commands to receive user input and turn it into game moves. The game they are playing is a maze game, the user is in a maze and has to find the way out using the commands left, right, up, down, go, and stop. The issue they ran into is the delay between the user speaking and the input happening, this is not an issue we have to deal with because chess can be played at a different pace than their game, but the article does highlight the potential issues with input delays.

This article focuses more on how games can be changed to support more accessibility features [13]. They cover a large number of topics, from dynamic difficulty to adaptive game controls, and real-time audio captioning. This article talks about potential changes to games to help support these features but also talks about potential issues. There is a large cost of using AI-driven tools, there are privacy concerns with data collection, and there is a potential for the game to be over-reliant on AI features. If properly kept in check, this article talks about a potential future for accessible gaming.

6. CONCLUSIONS

If we had more time, we would have made more improvements to how the program listens for audio input. Right now, the program listens for the “space” key to be pressed, when it is pressed it starts to record audio, when it is pressed a second time then the recording stops. We would improve the system by removing the key presses entirely [15]. When the microphone hears audio over a certain threshold, it would start recording, when that audio drops below a different threshold, it would stop. It would take that audio and send that instead to completely remove having to press a key. If we were to continue, we would add some of our own custom UI elements instead of using mostly default UI or what already existed inside the project [14]. If we had to restart we would first focus on improving the speech to commands to try and make it fluid to use, the speech to commands was one of the later features that was built.

REFERENCES

[1] Arık, Sercan Ö., et al. "Deep voice: Real-time neural text-to-speech." International conference on machine learning. PMLR, 2017.

[2] Keating, Elizabeth, and R. Neill Hadder. "Sensory impairment." Annual Review of Anthropology 39.1 (2010): 115-129.

[3] Fischer, Mary E., et al. "Multiple sensory impairment and quality of life." Ophthalmic epidemiology 16.6 (2009): 346-353.

[4] 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.

[5] Wright, William, et al. "The Sandbox for analysis: concepts and methods." Proceedings of the SIGCHI conference on Human Factors in computing systems. 2006.

[6] Lv, Xiaoling, Minglu Zhang, and Hui Li. "Robot control based on voice command." 2008 IEEE International Conference on Automation and Logistics. IEEE, 2008.

[7] Trivedi, Ayushi, et al. "Speech to text and text to speech recognition systems-Areview." IOSR J. Comput. Eng 20.2 (2018): 36-43.

[8] Lui, Thomas KL, Chuan-Guo Guo, and Wai K. Leung. "Accuracy of artificial intelligence on histology prediction and detection of colorectal polyps: a systematic review and meta-analysis." Gastrointestinal endoscopy 92.1 (2020): 11-22.

[9] Salton, Gerald, and Anita Wong. "On the role of words and phrases in automatic text analysis." Computers and the Humanities (1976): 69-87.

[10] Pilueta, Nino U., et al. "Chessbot: A voice-controlled chess board with self-moving pieces." AIP Conference Proceedings. Vol. 2502. No. 1. AIP Publishing, 2022.

[11] Anthes, Christoph, et al. "State of the art of virtual reality technology." 2016 IEEE aerospace conference. IEEE, 2016.

[12] Rossiou, Eleni, and Spyros Papadakis. "Applying Online Multiplayer Educational Games based on Generic Shells to Enhance Learning of Recursive Algorithms: Students' Preliminary Results." Proceedings of the 2nd European Conference on Games Based Learning. Vol. 373. 2008.

[13] Hamari, Juho, Jonna Koivisto, and Harri Sarsa. "Does gamification work?--a literature review of empirical studies on gamification." 2014 47th Hawaii international conference on system sciences. Ieee, 2014.
[14] Delgado, Antonio, et al. "Reusing UI elements with model-based user interface development." International Journal of Human-Computer Studies 86 (2016): 48-62.
[15] Verwey, Willem B. "A forthcoming key press can be selected while earlier ones are executed." Journal of motor behavior 27.3 (1995): 275-284.