Final Report

Accessible Chess

Kyle Davey
Ojasvi DSilva
David Thornton
Timothy VanSlyke

1. Problem Statement

Objective

The Accessible Chess system's goal is to allow a user to play a game of chess using voice commands. The system will be able to recognize voice commands from a user and act accordingly to those commands. At the same the system should audibly give the user all the information regarding the board so the user doesn't have to actually see the board to be able to play a game of chess.

Rationale

People with visual impairments cannot normally play chess games on a computer because they cannot see the board to make moves. The Accessible Chess system will allow people with visual impairments to play chess by accepting voice commands and using these commands to perform functions that those with visual impairments would not normally be able to do.

Existing Systems

There are currently no existing systems that solve this problem. There are chess programs that allow users to play chess in an application and there are also voice assistants that allow users to command their computer to perform actions using their voice. However, there is no current application that integrates these two types of systems to allow users to play chess using voice commands.

Proposed System

The proposed Accessible Chess system will use a voice recognition system and a game engine that will allow users to play games of chess through voice commands. The voice recognition system will use Google Speech to Text to turn speech into text and Dialogflow to convert the text into a command to execute. The system will accept and perform the command accordingly and notify the user when it has done so. For users without impairments, the system will use the open source Cutechess GUI to display the board and will allow users to make point and click commands using the GUI.

2. Team Details and Plan of Action

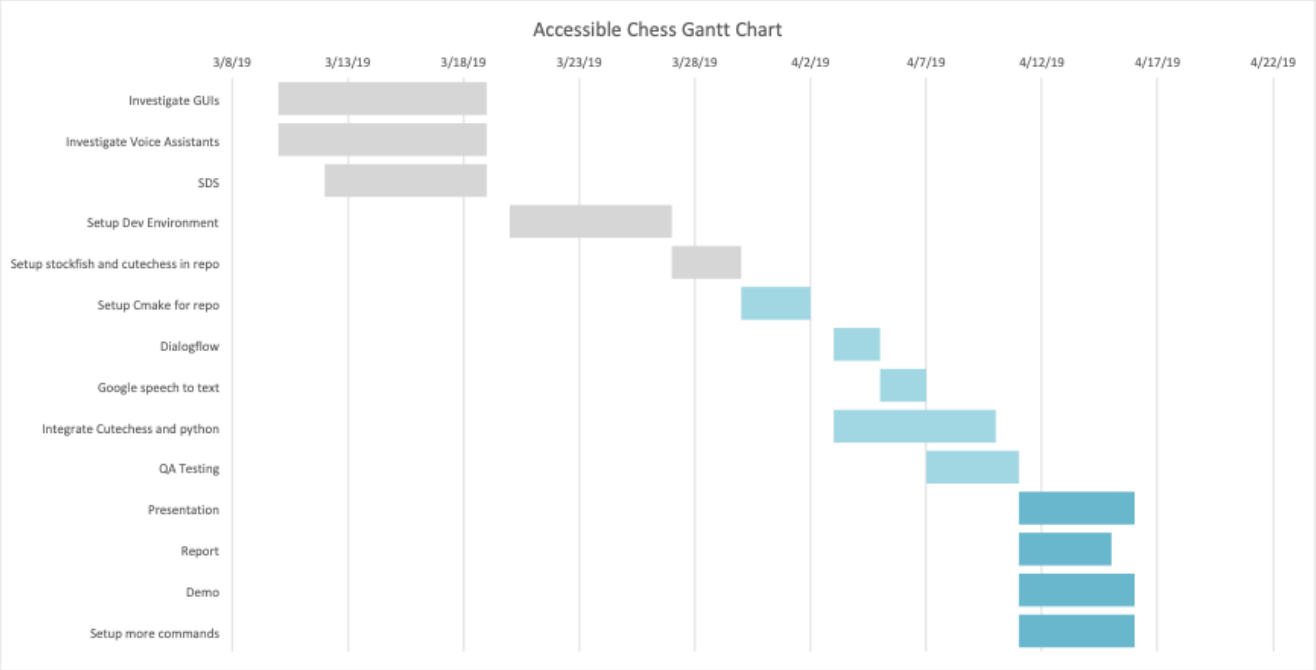
Team Roles

Kyle Davey - QA Testing
Ojasvi DSilva - Documents
David Thornton - Voice assistant and Dialogflow
Timothy VanSlyke - C++ to Python interpreter, Voice assistant and Dialogflow

Action Plans

TASK NAME	ASSIGNED TO	START DATE	DUE DATE	DURATION	% DONE	DESCRIPTION	PRIORITY	SPRINT/ MILESTONE
Investigate GUIs	David	3/10/19	3/19/19	9	100	Check for open source GUIs	1	Phase 1
Investigate Voice Assistants	David, Ojasvi, Kyle	3/10/19	3/19/19	9	100	Figure out possible voice assistants	1	Phase 1
SDS	Everyone	3/12/19	3/19/19	7	100	SDS Document	1	Phase 1
Setup Dev Environment	Everyone	3/20/19	3/27/19	7	100	Setup basic dev environment	1	Phase 2
Setup stockfish and cutechess in repo	David	3/27/19	3/30/19	3	100	Setup stockfish and cutechess	1	Phase 2
Setup Cmake for repo	Timothy	3/30/19	4/2/19	3	100	Use Cmake to build project	1	Phase 2
Dialogflow	David	4/3/19	4/5/19	2	100	Setup intents	1	Phase 3

Google speech to text	David	4/5/19	4/7/19	2	100	Speech to text	1	Phase 3
Integrate Cutechess and python	Timothy	4/3/19	4/10/19	7	100	Allow Cutechess to use python	1	Phase 3
QA Testing	Kyle	4/7/19	4/11/19	4	100	Make sure voice works	1	Phase 3
Presentation	Everyone	4/11/19	4/16/19	5	90	Final presentation	1	Phase 4
Report	David, Ojasvi, Kyle	4/11/19	4/15/19	4	100	Final report	1	Phase 4
Demo	David, Timothy	4/11/19	4/16/19	5	90	Demo for final presentation	1	Phase 4
Setup more commands	Timothy	4/11/19	4/16/19	5	90	Add more commands	1	Phase 4



3. Requirement Analysis (From SRS)

3.1. Introduction

3.1.1 Purpose of this Document

The purpose of this document is to give a comprehensive description of the requirements for the “Accessible Chess” software. This document will show the purpose and complete plan for the design and development of the system. It will explain requirements, both functional and nonfunctional, and how the system interacts with users. The target audiences of this document are the customer to get its approval as well as developers in charge of implementing the system.

3.1.2 Scope of the Development Process

The “Accessible Chess” software is an application that allows people who are visually impaired to play chess without having to see the chess board. The distinct feature of software is that it allows users to play the game solely using vocal commands instead of the standard point and click. The user can uses vocal commands to to create new games, move pieces and find out all the information on the board such as the location of pieces or who has taken the most pieces. The benefits of this is that people who would normally be unable to play chess in the past due their disability are now able to do so. This opens up the game to a wider range of players. This software can also be used by other who simply wish to have a more convenient way of playing chess.

3.1.3 Overview of Document

The following document will detail the various components and characteristics of the “Accessible Chess” software. In the second section it will describe the characteristics of the user, the functional requirements, the data requirements and a product perspective. It will also discuss some of general constraints, assumptions and dependencies we had with this software. In section 3, the document will go into more detail of the requirements of this software such as external interfaces, functions and performance requirements.

3.2 General Description

3.2.1 User Characteristics

Users of the software include people with and without visual impairments, and the application is designed to support both types of users. Thinking about users who are visually impaired, the system allows users to play a match of chess entirely using voice commands. Accepted voice commands include being able to get the current game state, move piece, undo move, create new game, query attackers. The system also supports controls to play a standard game without the use of voice commands.

3.2.2 Product Perspective

“Accessible Chess” is a standalone product that makes use of the functionality featured in the stockfish chess engine in conjunction with a voice recognition system. The voice recognition system is composed of Snowboy Hotword Detection, Google Cloud Speech to Text API, and Dialogflow.

3.2.3 Overview of Functional Requirements

The user will be able to:

1. Use vocal commands and point-and-click commands
2. Move a chess piece
3. Create a new game
4. Cancel the creation of a new game
5. Resign a game
6. Load a game
7. Save a game
8. Select a side to play (white or black)
9. Query attackers using voice commands
10. Query pieces using voice commands
11. Query materials using voice commands
12. Be notified if a command is not valid
13. Receive an echo of each command accepted
14. Receive an audible announcement of each move made

3.2.4 Overview of Data Requirements

Users are able to save their current game and then load the previous game when needed.

3.2.5 General Constraints, Assumptions, Dependencies, Guidelines

Constraints may include linking the Stockfish chess engine, which is written in C++, to the voice recognition system, which is written in Python. Other constraints include the accuracy of voice commands and making sure the application can properly interpret various commands or act appropriately if the voice command can not be interpreted.

The application will support the following operating systems:

- macOS
- Windows (32-bit and 64-bit)
- Linux (32-bit and 64-bit)

3.2.6 User View of Product Use

For a visually impaired user, they will start the game by issuing the appropriate vocal command. The game will then audibly ask the user which side they would like to play on before starting the match. Once the match has been loaded the user will be able speak the various commands that enable progression of the match. Throughout the duration of the match the user will be able to speak commands to hear the game state and current attacking piece positions. The user will also have the ability to issue commands to save or exit the current game.

Users not using the voice command features will be greeted with the Cutechess GUI. The Cutechess GUI comes with many point and click features that will allow users without impairments to play chess like they normally do.

3.3 Specific Requirements

3.3.1 External Interface Requirements

A user of the system has three primary options for interfacing with the system:

1. GUI Control - Like most conventional chess applications, the system shall provide a graphical interface for controlling a session.
2. Voice Control - The system shall provide an audio/vocal interface via the voice recognition system composed of Snowboy Hotword Detection, Google Cloud Speech to Text API, and Dialogflow.
3. Text Control - The system shall provide a text-based interface in which the user may type commands which are treated identically to vocal commands.

On software startup the user is presented with the graphical interface with a new game started by default. The GUI offers many buttons such as save the current game, load a previous game, move a piece, start a new game, and resign the current game. Pressing the “create new game” button presents the user with GUI controls to modify parameters such as which color the user wants to play (white or black) and whether it is a one player game or a two player game. Pressing the “load game” button opens a predetermined folder where saved game files reside.

Once a game is started, a graphical representation of the game board is presented. If the game is timed, a timer is shown alongside the board. Additionally, a graphical representation of taken material is shown for both players. A game history is also displayed. This information is also available to the user the vocal and text interfaces. Users may issue commands to query positions on the board or materials.

Using point-and-click or drag-and-drop mouse actions, the user may move pieces on the board during their turn. The same functionality is available via vocal commands.

3.3.2 Detailed Description of Functional Requirements

3.3.2.1 Accept User Command

- Purpose: Allow users to navigate the software using vocal and point and click commands.
- Trigger: User issues command either through speaking or using a mouse.
- Result: Calls whichever command the user wants
- Exception: Thrown if the command is not valid or not possible to be executed.

3.3.2.2 Move Piece

- Purpose: Move whatever chess piece the user chooses to move.

- Trigger: User issues command to move piece either vocally or using a mouse.
- Result: Chess piece is moved to the desired location
- Exception: Thrown if piece is not on board or piece cannot move to chosen space

3.3.2.3 Create a Game

- Purpose: Create a new game for the user.
- Trigger: User issues command to create a game either vocally or using a mouse.
- Result: New game is created
- Exception: Thrown if there is a game already active.

3.3.2.4 Cancel the Creation of a New Game

- Purpose: Cancel the new game creation flow
- Trigger: User has started the new game flow and does not want to create a new game
- Result: New game is not created
- Exception: If there is not a new game being created

3.3.2.4 Resign Game

- Purpose: Allow user to concede a started game.
- Trigger: User issues command to resign either vocally or using a mouse.
- Result: The current game is ended and the user issuing the command loses the game.
- Exception: Thrown if there is not an existing game

3.3.2.5 Load Game

- Purpose: Allow the user load a game that they had previously saved.
- Trigger: User issues command to load a game either vocally or using a mouse.
- Result: Exits the current game and loads the saved game state
- Exception: Thrown if there is no saved game

3.3.2.6 Save Game

- Purpose: Allow the user save a game they are currently playing.
- Trigger: User issues command to save a game either vocally or using a mouse.
- Result: The current game is saved
- Exception: Thrown if there is no current game

3.3.2.7 Present Material

- Purpose: Inform the user, vocally or otherwise, of the material counts for each player in the currently-active game.
- Trigger: User issues a command requesting material counts.
- Result: Voice assistant, GUI, or text interface displays or vocalizes material counts for each player.

- Exception: Thrown when there is no currently-active game.

3.3.2.8 Present Position Information

- Purpose: Inform the user, vocally or otherwise, of the state of positions on the game board.
- Trigger: User issues a command requesting board position information.
- Result: Voice assistant, GUI, or text interface displays or vocalizes the state of the requested position.
- Exception: Thrown when there is no currently-active game.

3.3.2.9 Present Attackers Information

- Inform the use vocally or by text, all the pieces attack a certain square on the board.
- Trigger: User issues a command requesting information on all the pieces attack a specific square on the board.
- Result: Voice assistant, GUI, or text interface displays or vocalizes a list of pieces attacking the specified square.
- Exception: Thrown when there is no currently-active game

3.3.2.10 Notify Invalid Command

- Purpose: Inform the user, vocally or otherwise, if an invalid request or command is received.
- Trigger: User issues a command that is invalid in some way.
- Result: Voice assistant, GUI, or text interface displays or vocalizes that the given command is invalid.
- Exception: None.

3.3.2.11 Echo Each Command Accepted

- Purpose: Echo or repeat a command made by the user vocally or otherwise that was accepted.
- Trigger: User issues command vocally or textually
- Result: Voice Assistant echos the command made by the user.
- Exception: The command is not accepted.

3.3.3 Performance Requirements

The system shall have a delay of no more than 1 second on all inputs which do not invoke the voice assistant or the chess engine. This includes new game creation, loading saved games, and moving pieces in a game. An exception to this requirement may be allowed for loading saved games when hard drive contention is sufficiently high that it impedes reading saved game files in a timely manner.

Vocal commands issued to the systems shall be audibly acknowledged in less than three seconds. Actual execution of vocal commands issued to the system may exceed this threshold as resources permit. In particular, the voice assistant may be hindered by slow network connection or contention with other processes of system resources such as memory CPU time.

Invocations of the chess engine shall take no longer than ten seconds. While pondering, the chess engine may take an arbitrary amount of time to compute optimal moves. Note that this does not affect other timing requirements; a vocal command which initiates pondering must be acknowledged within a one-second timeframe.

3.3.4 Quality Attributes

N/A

3.4. Other Requirements

N/A

4. System Design (From SDS)

4.1 Introduction

4.1.1 Purpose of this Document

The purpose of this document is to give a comprehensive description of the design for the “Accessible Chess” software. This document will show the purpose and complete plan for the design and development of the system. It will explain the scope of the project, the different requirements, the data design, the system architecture design, and the components that make up the system. The target audiences of this document are the developers and coders who will be implementing the design as well as the testers who will need to verify that the software does what it is supposed to do.

4.1.2 Scope of the Development Project

The “Accessible Chess” software is an application that allows people who are visually impaired to play chess without having to see the chess board. The distinct feature of software is that it allows users to play the game solely using vocal commands instead of the standard point and click. The user can uses vocal commands to to create new games, move pieces and find out all the information on the board such as the location of pieces or who has taken the most pieces. The benefits of this is that people who would normally be unable to play chess in the past due their disability are now able to do so. This opens up the game to a wider range of players. This software can also be used by other who simply wish to have a more convenient way of playing chess.

4.1.3 Major Software Requirements

The application will support the following operating systems:

- macOS
- Windows (32-bit and 64-bit)
- Linux (32-bit and 64-bit)

The application will use the Stockfish chess engine (open source), Cutechess GUI (open source), and a voice recognition system.

4.1.4 Design Constraints, Limitations

This section will describe the constraints and limitations of our design, including the commercial, usability, performance and integration constraints and limitations.

4.1.5.1 Design Constraints

The system shall have a delay of no more than 1s on all inputs which do not invoke the voice assistant or the chess engine. This includes new game creation, loading saved games, and moving pieces in a game. An exception to this requirement may be allowed for loading saved games when hard drive contention is sufficiently high that it impedes reading saved game files in a timely manner.

Vocal commands issued to the systems shall be audibly acknowledged in less than five seconds. Actual execution of vocal commands issued to the system may exceed this threshold as resources permit. In particular, the voice assistant may be hindered by slow network connection or contention with other processes of system resources such as memory CPU time.

Invocations of the chess engine shall take no longer than ten seconds. While pondering, the chess engine may take an arbitrary amount of time to compute optimal moves. Note that this does not affect other timing requirements; a vocal command which initiates pondering must be acknowledged within a one-second timeframe.

4.1.5.2 Limitations

The limitation of this design is time, since there is a small amount of time left in the semester. Since this is the case, requirements may need to be modified as the deadline approaches.

4.1.6 Changes to Requirements

There are no changes to our requirements.

4.1.7 Overview of Document

The rest of the document will detail the various components that make up the design of the “Accessible Chess” software. The second section will describe the data design of the system, including the data objects, data structures, file structures, and database structures. The third section will describe the system architecture including the modules, components, structure, and relationships between modules. The fourth section will give a detailed description of the components that were introduced in section three. The fifth section will describe the interface design of the system.

4.1.7 Design Goals

- Performance: At most a five second response time for each command.
- Usability: Supported by the three main operating systems (macOS, Windows, Linux).
- Availability: System can be started at any time of the day.

4.2. Data Design

4.2.1 Data Objects and Resultant Data Structures

4.2.1.1 Move

Purpose: Allow the chess engine to determine if moving from one space to another is the best move
Fields: Two BoardPosition representing the start and end positions

4.2.1.2 BoardPosition

Purpose: Part of a move, containing the row and the column of a space on the chess board
Fields: Two int representing the column and row of the board

4.2.1.3 Board

Purpose: An 8x8 container containing pieces located on different spaces
Fields: A 2D array containing ChessPieces

4.2.1.4 MaterialCounter

Purpose: Counts the number of pieces left for each side
Fields: A Board

4.2.1.5 ChessPiece

Purpose: A unit on the board that can move around the board
Fields: A ChessPieceKind and a Color

4.2.1.6 ChessPieceKind

Purpose: The type of chess piece, which defines the different movement a piece has
Fields: None

4.2.1.7 Color

Purpose: Represents which player is in control of the piece

Fields: None

4.2.2 File and Database Structures

4.2.2.1 External File Structure

When a game is saved, a file will be created and saved in a folder that will contain saved files. These saved files can be loaded at a later time. These files will contain the positions of pieces on the board, the color of the side whose turn it is, and other data needed to load game information at a later time.

4.2.2.2 Global Data

N/A

4.2.2.3 File and Data Cross Reference

- Move is composed by two BoardPosition
- Board class is composed of multiple ChessPieces
- ChessPiece has both a Color and ChessPieceKind

4.3. System Architecture Description

4.3.1 Overview of Modules and Components

There are three modules in our system:

1. GUI module
2. Voice recognition module
3. Cutechess module

The GUI module and Voice recognition modules both serve as parts of the view module. Since the Cutechess module is open source, it has the model and controller modules already included.

4.3.1.1 GUI Module

Functionality: Displays the chess game to the user and allows them to make point and click commands

4.3.1.2 Voice Recognition Module

Functionality: Accepts vocal commands from the user

4.3.1.3 Cutechess Module

Functionality: Controls the game board and edits it as needed based on received commands

4.3.2 Structure and Relationships

The GUI Module and the voice recognition modules communicate with the Cutechess module by sending it commands. The Cutechess module then edits the game board and redisplays the board as needed.

4.4. Detailed Description of Components

4.4.1.1 Description of Voice Recognition Component

Identification
Voice Recognition Component in the Voice Recognition module

Type
Class

Purpose
Allows the user to communicate with the system using speech.

Function
- Takes in a vocal command and sends it to the Cutechess module
- Notifies the user of changes to the board
- Does not store any data or modify any data

Subordinates/ Modules used
This component does not use other modules, but it sends data to the Cutechess module

Dependencies
This component depends on the Cutechess module to execute commands.

Resources
Memory in order to run python scripts.

Processing
Takes in speech from the user, converts the speech to text, finds the correct command, and sends the command to the Cutechess module.

Data
Sends the command name to the Cutechess module along with any arguments needed with that command.

4.4.2 Description of GUI Component

Identification
GUI

Type
Class

Purpose
Shows the user the chess board and takes in point and click commands.

Function
Takes in point and click commands and sends it to the Cutechess module. Does not store or modify data.

Subordinates/ Modules used
This component sends data to the Cutechess module

Dependencies
Does not depend on any other modules

Resources
Requires memory to display the board and uses I/O devices.

Processing
Takes in a point and click command and sends it to the Cutechess Module

Data
Sends the command name to the Cutechess module along with any arguments needed with that command.

4.4.3 Description of Cutechess Component

Identification
Cutechess

Type
Class

Purpose
Modifies the chess board based on the command given to it.

Function
Receives commands from the Voice Recognition module and the GUI module and executes those commands.

Subordinates/ Modules used

Receives data from the Voice Recognition Module and the GUI Module

Dependencies
Depends on the GUI module and the Voice Recognition module

Resources
Requires memory to store board state and execute methods on the board.

Processing
Takes in data from the Voice Recognition module and the GUI module and performs the correct command using that data and the resulting arguments

Data
The data received is a command and the arguments needed with the commands.

4.5. Interface Design

For a visually impaired user, they will start the game by issuing the appropriate vocal command. The game will then audibly ask the user which side they would like to play on before starting the match. Once the match has been loaded the user will be able speak the various commands that enable progression of the match. Throughout the duration of the match the user will be able to speak commands to hear the game state and current attacking piece positions. The user will also have the ability to issue commands to save or exit the current game.

Users not using the voice command features will be greeted with an easy-to-follow UI. The UI will show a chess board and will allow the user to make point and click commands like a regular chess application.

5. Test Document

5.1. Create New Game

Name: CreateNewGame
Entry Condition: The default game is active.
Flow of Events:
1. Attempt to create a new game; ensure an exception is thrown.
2. Forfeit the current game.
3. Attempt to create a new game; ensure that no exception is thrown.
4. Attempt to create a new game; ensure an exception is thrown.
5. Set the Black player to CPU.
6. Set the Black player to Human.
7. Set the White player to CPU.
8. Accept.
Exit Condition: A new game is currently active. The black player is a human and the white player is a CPU.

5.2. Choose Side

Name: ChooseSide
Entry Condition: The default game is active.
Flow of Events:
1. Forfeit the current game.
2. Attempt to create a new game; ensure that no exception is thrown.
4. Set the Black player to CPU.
5. Set the White player to CPU.
6. Set the White player to Human.
7. Accept.
Exit Condition: A new game is currently active. The white player is a human and the black player is a CPU.

5.3. Forfeit Game

Name: ForfeitGame
Entry Condition: The default game is active.
Flow of Events:
1. Attempt to forfeit the game; ensure no exception is thrown.
2. Attempt to forfeit the game; ensure an exception is thrown.
3. Create a new game; ensure no exception is thrown.
4. Attempt to forfeit the game; ensure no exception is thrown.

5. Attempt to forfeit the game; ensure an exception is thrown.
6. Create a new game; ensure no exception is thrown.
Exit Condition: There is no active game

5.4. Make Move

Name: Make Move
Entry Condition: The default game is active.
Flow of Events:
1. Move A2 A4; ensure no exception is thrown.
2. Move A2 A4; ensure an exception is thrown.
3. Move H7 H5; ensure no exception is thrown.
4. Move B2 B3; ensure no exception is thrown.
3. Move H8 H6; ensure no exception is thrown.
Exit Condition: There is an active game with a white pawn on A4, a white pawn on B3, a black pawn on H5 and a black rook on H6.

5.5. Cancel New Game

Name: CancelNewGame
Entry Condition: The default game is active.
Flow of Events:
1. Forfeit the current game.
2. Attempt to create a new game; ensure that no exception is thrown.
3. Set the Black player to CPU.
4. Set the Black player to Human.
5. Set the White player to CPU.
6. Cancel new game; ensure that no exception is thrown.
Exit Condition: No game is currently active.

5.6. Save and Load Game

Name: SaveAndLoadGame
Entry Condition: The default game is active.
Flow of Events:
1. Move A2 A4; ensure no exception is thrown.
2. Move H7 H5; ensure no exception is thrown.
3. Move B2 B3; ensure no exception is thrown.
4. Move H8 H6; ensure no exception is thrown.
5. Save the current game.
6. Attempt to create a new game; ensure that no exception is thrown.
7. Load the saved game.
8. Finish creating a new game.

Exit Condition: There is an active game with a white pawn on A4, a white pawn on B3, a black pawn on H5 and a black rook on H6.

5.7. Query Attackers

Name: QueryAttackers
Entry Condition: The default game is active.
Flow of Events:
1. Move A2 A4; ensure no exception is thrown.
2. Move H7 H5; ensure no exception is thrown.
3. Move D2 D3; ensure no exception is thrown.
4. Move H8 H6; ensure no exception is thrown.
5. Query attackers for H6; ensure black pawn on G7, black knight on G8, and white bishop on C1 are listed as attackers.
Exit Condition: There is an active game with a white pawn on A4, a white pawn on C3, a black pawn on H5 and a black rook on H6.

5.8. Query Material

Name: QueryMaterial
Entry Condition: The default game is active.
Flow of Events:
1. Move D2 D4; ensure no exception is thrown.
2. Move E7 E5; ensure no exception is thrown.
3. Move D4 E5; ensure no exception is thrown.
4. Query material; ensure a black pawn is listed.
Exit Condition: There is an active game with a white pawn on E5, and no black E pawn.

6. User Guide

There are few components you need to download and install on your computer before being able to use the Accessible Chess Software. Specifically you will first need to download and install Python 3, Cmake and QT. The following Python libraries are also needed:

- Dialogflow
- Snowboy
- Pyaudio
- Pyttsx3
- Pyobjc

Once you have those software installed you must build the project by running “cmake”. Then you can then start the software by clicking the executable file AccessibleChess.exe or by running it from the command line by using “./cutechess/cutechess”. You can start speaking to the voice assistant by simply saying the hotword, which is currently “Gambit”.

To start a new game you just say “New Game” or “Create Game.” You can select which side you want to be on by saying “Set White to Human” or “Set White to CPU”. Once you are don you just say “ Done creating new game” or if you wish to cancel the creation of a new game you can say “ Cancel New Game”. Alternatively you can load a previously saved game by saying “Load Game”.

Once you are in a game you have several voice command options. To move a piece you say “move” and then say the square the piece is currently and then the square you wish to move the piece to. For example “Move C1 C2” or “Move A1 A2”. You can also query to voice assistant on the current status of the board. You can query what pieces are currently attacking a specific square on the board by saying “ Attackers for position A2” or just “Attackers A2”. You can query how many pieces have been captured by each side by asking “What is the taken material for each side?” And finally you can query what piece is on a specific square by asking “ What is the piece at A4?”.

At any point during the game you can save the current game or forfeit the game. To save the game you say “Save Game” and to forfeit the game you say “Forfeit Game”.

7. Glossary

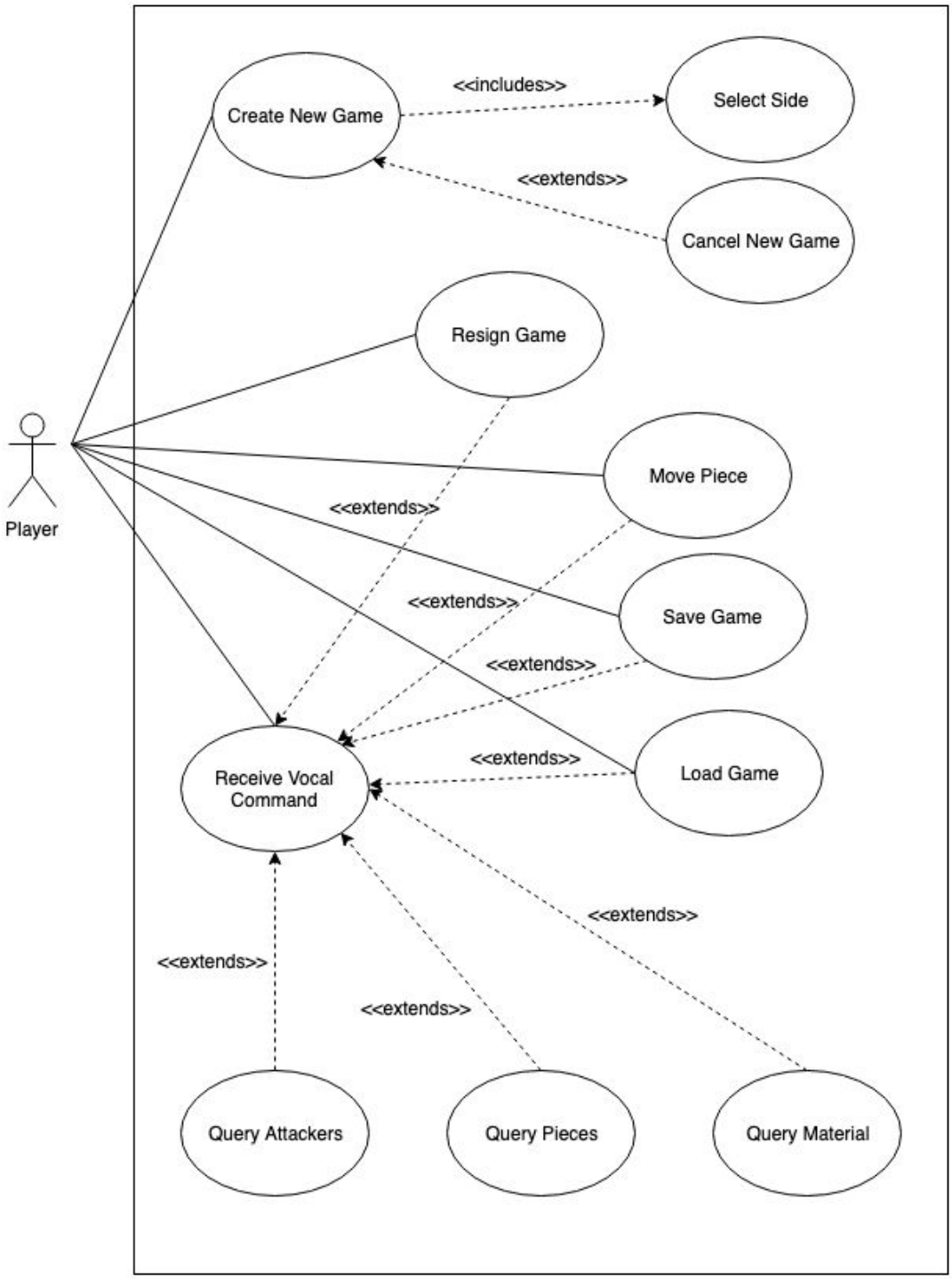
Term	Definition
Capture	An action resulting in a chess piece moving from one space on the board to another space on the board occupied by a chess piece of the opposing side. The opposing chess piece is removed from the game.
Check	A condition when a player's king piece is under threat of capture on their opponent's next turn
Checkmate	A condition when a player's king piece cannot escape from the "Check" condition. The player who put the opponent in Checkmate wins the game.
Chess Move	An action resulting in a chess piece moving from one space on the board to another space on the board
Chess Piece	A single unit in a chess match
Command	A generic order to the software, either spoken or using a mouse
Game	A single chess match
User	A person who interacts with the "Accessible Chess" software
Visual Impairment	A condition resulting in a user being unable to clearly see the position of pieces on a chess board
Vocal Command	A spoken order to the software

8. References

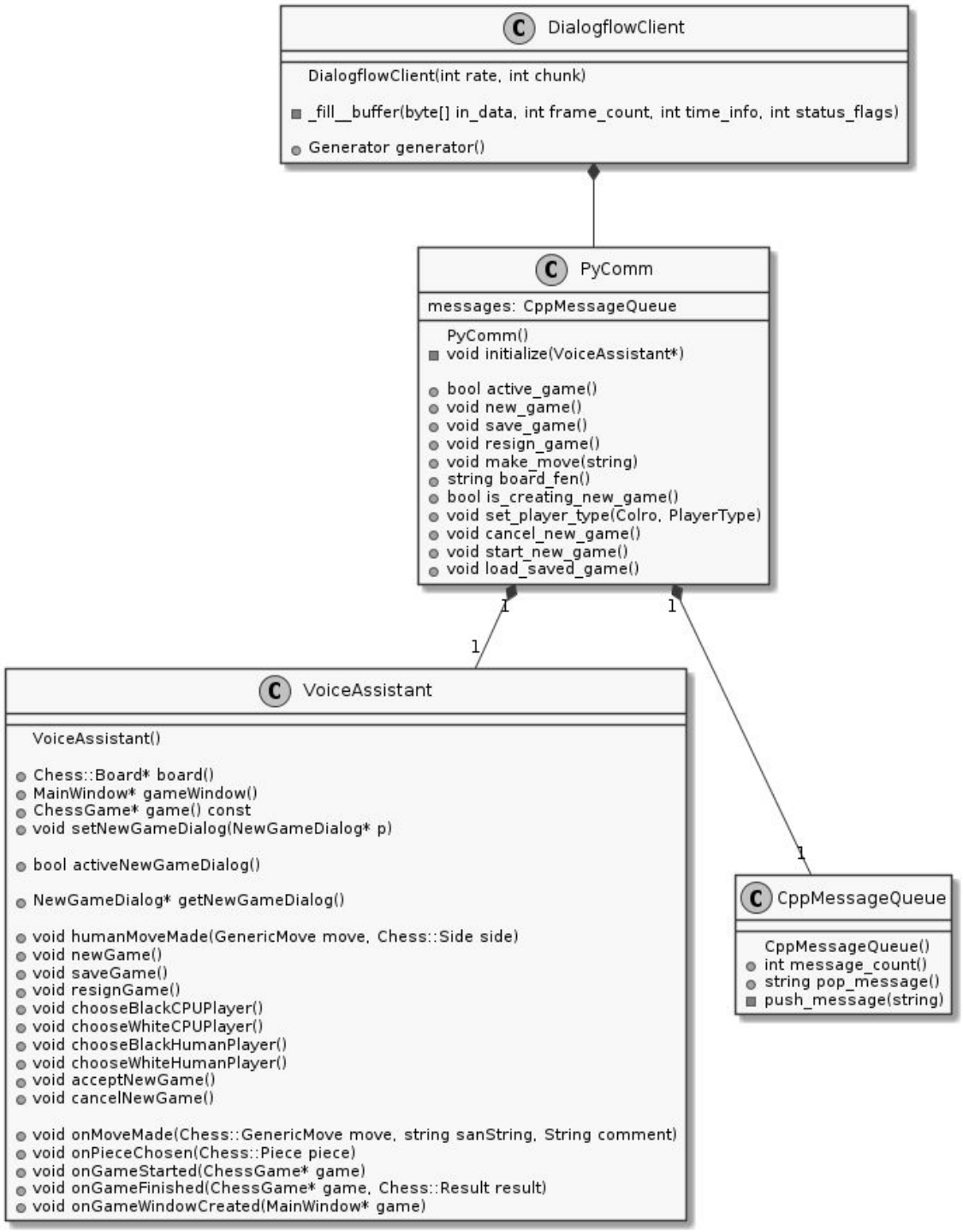
Cutechess - https://github.com/cutechess/cutechess (Source code)
Dialogflow - https://dialogflow.com/docs/sdks?authuser=3 (Documentation and Console)
Stockfish - https://stockfishchess.org/download/ (source code)
Google Speech to Text API - https://cloud.google.com/speech-to-text/docs/reference/libraries (Documentation)
Snowboy - https://github.com/kitt-ai/snowboy (source code and how to build)
QT (https://www.qt.io/download)

9. Appendix

Appendix A: Use Case Diagram

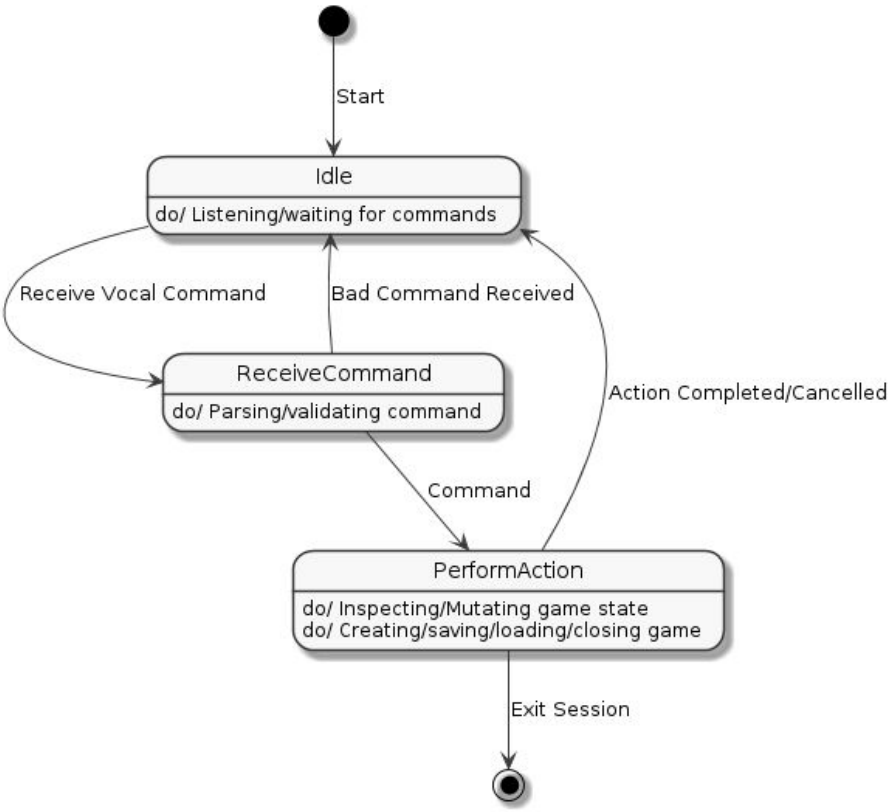


Appendix B: Class Diagram

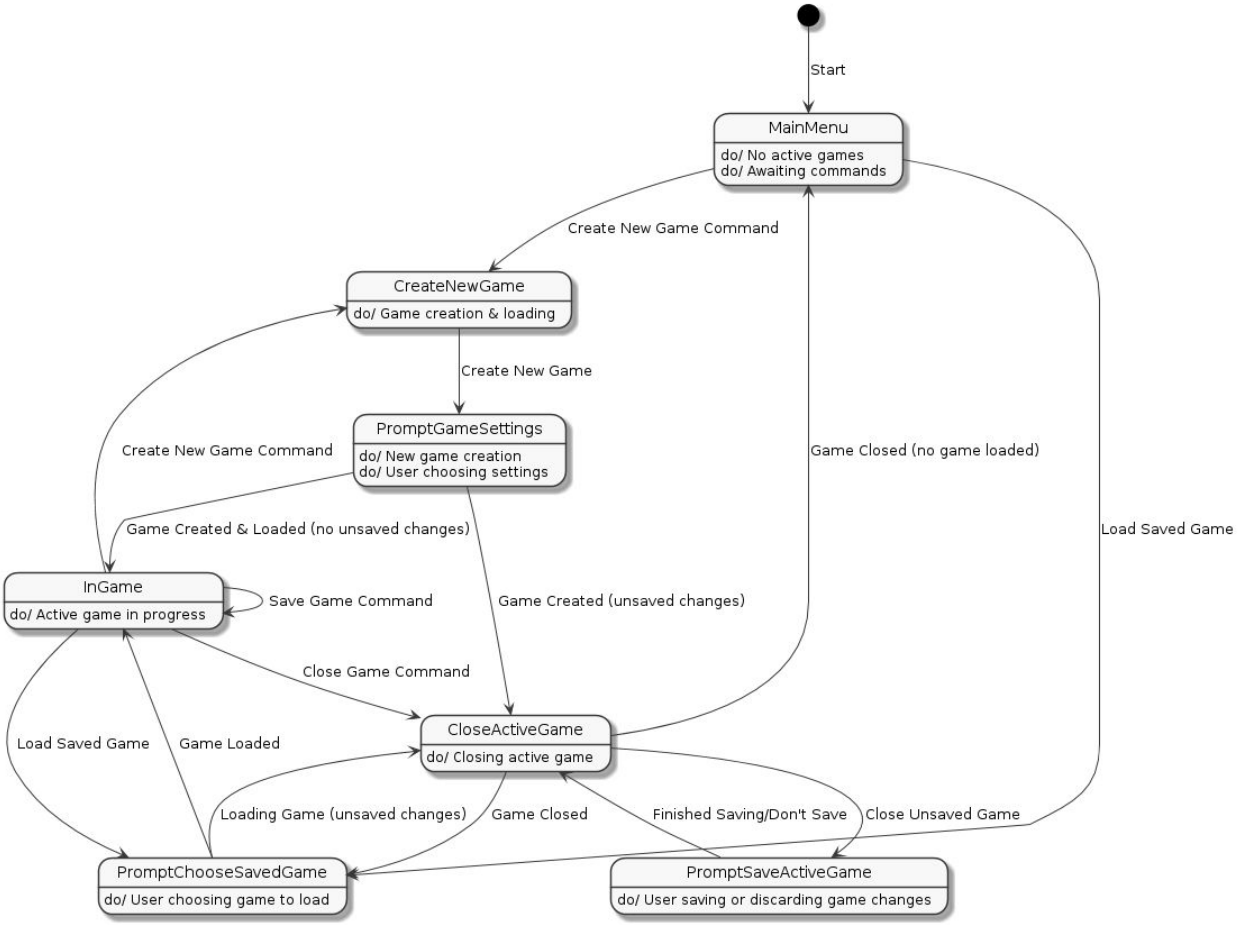


Appendix C: State Transition Diagrams

Appendix C.1: Session State Transition Diagram

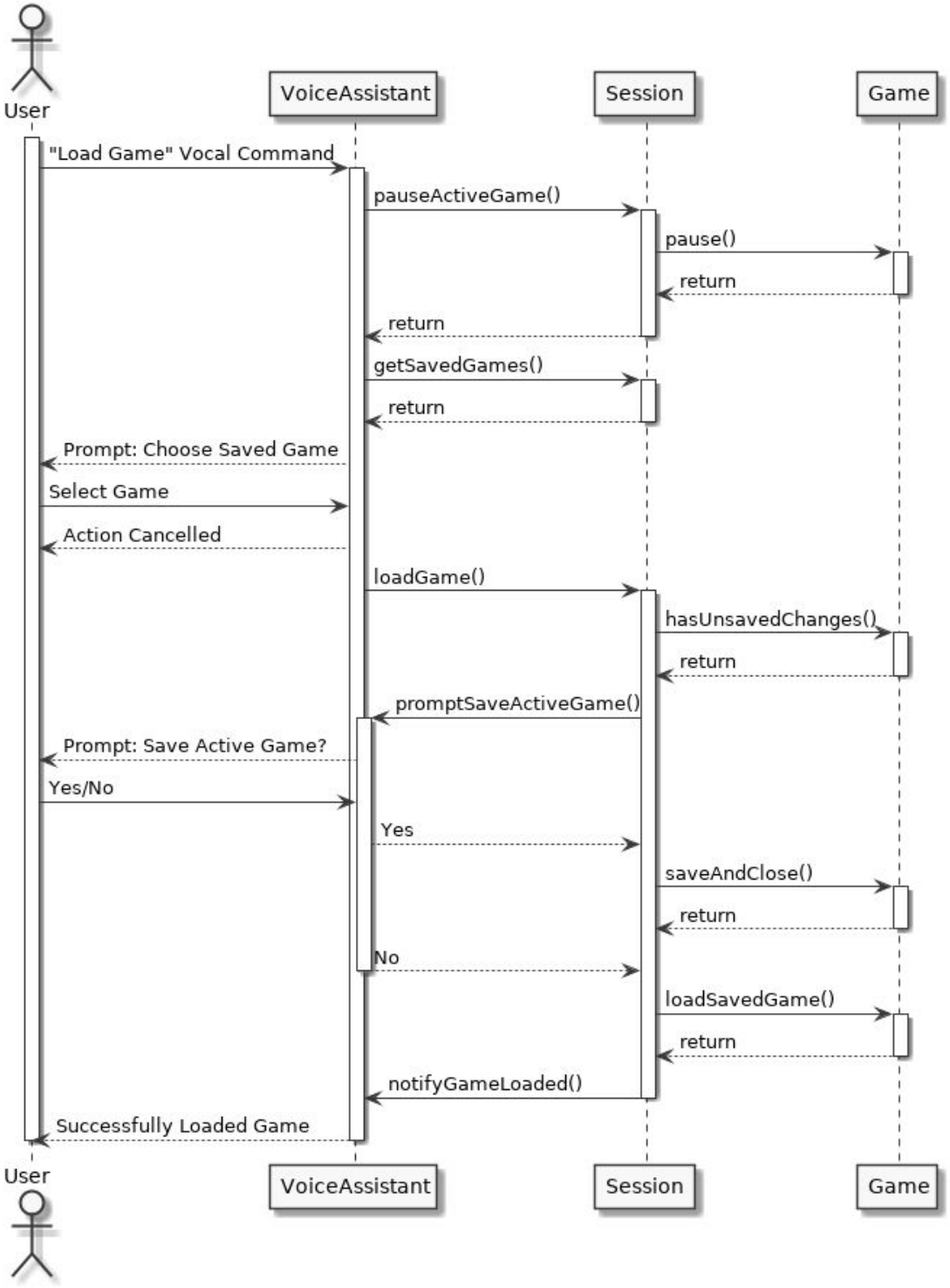


Appendix C.2: Top Level Command State Transition Diagram



Appendix D: Sequence Diagrams

Appendix D.1: Load Game Sequence Diagram



Appendix E: Use Cases

Appendix E.1: Receive Vocal Command Use Case

Use Case Identification and History			
Use Case ID	ReceiveVocalCommand		
Use Case Name	Receive Vocal Command	**Version Number**	
End Objective	Accept a vocal command from the user and go to the correct next use case		
Created By		**On (date):**	
Last Update By		**On (date):**	
Approved By		**On (date):**	
User/Actor	Player		
Business Owner Name		**Contact Details**	
Trigger	Player issues a vocal command		
Frequency of Use	This use case can be invoked an arbitrary number of times per game.		

Preconditions

Basic Flow		
Step	**User Actions**	**System Actions**
1	User issues a voice command	System determines what command was issued
2		System performs action

Exception Flow		
Step	**User Actions**	**System Actions**
1	User issues voice command	System notifies user that command was invalid

Post Conditions

Includes and Extension Points
Extended by: ● QueryAttackers ● QueryPiece ● QueryMaterial ● UndoMove ● CreateNewGame ● ResignGame ● MovePiece ● ExitGame ● LoadGame

Special Requirements

Business Rules

Other Notes
This use case describes an accessibility feature for visually impaired players.

Appendix E.2: Query Attackers Use Case

Use Case Identification and History			
Use Case ID	QueryAttackers		
Use Case Name	Query Attackers	**Version Number**	
End Objective	Inform the player of which pieces on the board are currently attacking a position.		
Created By		**On (date):**	
Last Update By		**On (date):**	
Approved By		**On (date):**	
User/Actor	Player		
Business Owner Name		**Contact Details**	
Trigger	The player issues a vocal command to query which pieces are attacking a position.		
Frequency of Use	The command may be issued an arbitrary number of times during a game.		

Preconditions
● There is a game currently active.

Basic Flow		
Step	**User Actions**	**System Actions**
1	The player issues a vocal command to query which pieces are attacking a position.	The system audibly lists each piece, along with its position and color, that is currently attacking the position.

Exception Flow		
Step	**User Actions**	**System Actions**
1	The player issues a vocal command to query which pieces are attacking a position while there is no game active.	The system informs the user that there is no game active.

Post Conditions
● User is then asked to make a move(Normal Flow) ● The system preserves its previous state (exceptional flow).

Includes and Extension Points
Extends: ReceiveVocalCommand

Special Requirements

Business Rules

Other Notes
This use case describes an accessibility feature for visually impaired players.

Appendix E.3: Query Piece Use Case

Use Case Identification and History

Use Case ID	QueryPiece		
Use Case Name	Query Piece	**Version Number**	
End Objective	Inform the player what if there is a piece on a square on the board and what said piece is. .		
Created By		**On (date):**	
Last Update By		**On (date):**	
Approved By		**On (date):**	
User/Actor	Player		
Business Owner Name		**Contact Details**	
Trigger	The player issues a vocal command to query a position on the board.		
Frequency of Use	The command may be issued an arbitrary number of times during a game.		

Preconditions
● There is a game currently active.

Basic Flow		
Step	**User Actions**	**System Actions**
1	The player issues a vocal command to query a position on the board.	The system verifies that a game is currently active and informs the user of what piece is currently located at the specified position, if there is one there.

Exception Flow

Step	User Actions	System Actions
1	The player issues a vocal command to query a position on the board while there is no active game.	The system audibly informs the player that there is no active game.

Post Conditions
● User is then asked to make a move(Normal Flow) ● The system preserves its previous state (exceptional flow).

Includes and Extension Points
Extends: ReceiveVocalCommand

Special Requirements

Business Rules

Other Notes
This use case describes an accessibility feature for visually impaired players.

Appendix E.4: Query Material Use Case

Use Case Identification and History	
Use Case ID	QueryMaterial

Use Case Name	Query Material	**Version Number**	
End Objective	Inform the user of what pieces have been taken and state which player has a material advantage and by how much.		
Created By		**On (date):**	
Last Update By		**On (date):**	
Approved By		**On (date):**	
User/Actor	Player		
Business Owner Name		**Contact Details**	
Trigger	The player issues a vocal command to query what pieces they have taken and what pieces they have lost.		
Frequency of Use	The command may be issued an arbitrary number of times during a game.		

Preconditions
● There is a game currently active. ● At least one piece in the game has been taken

Basic Flow		
Step	**User Actions**	**System Actions**
1	The player issues a vocal command to query the board on what pieces have been taken.	The system verifies that a game is currently active and informs the player which pieces they has taken and which pieces he has lost.

Exception Flow		
Step	**User Actions**	**System Actions**
1	The player issues a vocal	The system audibly informs

	command to query the board on what pieces have been taken when there is no active game.	the player that there is no active game.
2	The player issues a vocal command to query the board on what pieces have been taken, when all the pieces are still on the board.	The system verifies that a game is currently active and informs the player that all the pieces are still on the board.

Post Conditions
● User is then asked to make a move(Normal Flow) ● The system preserves its previous state (exceptional flow).

Includes and Extension Points
Extends: ReceiveVocalCommand

Special Requirements

Business Rules

Other Notes
This use case describes an accessibility feature for visually impaired players.

Appendix E.5: Create New Game Use Case

Use Case Identification and History	
Use Case ID	CreateNewGame

Use Case Name	Create New Game	**Version Number**	
End Objective	Begin a new game either against the engine, or against another player		
Created By		**On (date):**	
Last Update By		**On (date):**	
Approved By		**On (date):**	
User/Actor	Player		
Business Owner Name		**Contact Details**	
Trigger	Vocal command to start a new game, or new game button		
Frequency of Use	Once per game. Can occur an arbitrary number of times while the software is running		

Preconditions
A new game can be initiated at any time.

Basic Flow		
Step	**User Actions**	**System Actions**
1	User issues command to start new game.	The system halts the currently-active game, if any, and asks user to specify game options.
2	User issues commands to specify game options.	The system starts new game, discarding the state of the previous game.

Exception Flow

Step	User Actions	System Actions
1	User requests that the system abandon the new game.	The system discards the new game request and resumes the previous game, if any.

Post Conditions
● User is then asked to select side (normal flow). ● The system preserves its previous state (exceptional flow).

Includes and Extension Points
● Includes: SelectSide, EndGame ● Extends: ReceiveVocalCommand ● Extended by: CancelNewGame

Special Requirements
The system should only discard the currently-active game once the new game creation process has finished successfully.

Business Rules

Other Notes

Appendix E.6: Select Side Use Case

Use Case Identification and History			
Use Case ID	SelectSide		
Use Case Name	Select Side	**Version Number**	
End Objective	Select side of the board to play on		
Created By		**On (date):**	
Last Update By		**On (date):**	
Approved By		**On (date):**	
User/Actor	Player		
Business Owner Name		**Contact Details**	
Trigger	After issuing CreateNewGame command, user will select side to play on		
Frequency of Use	Occurs after each CreateNewGame		

Preconditions
● The user has confirmed they want to create a new game

Basic Flow		
Step	**User Actions**	**System Actions**
1	User selects one of the sides to play on	The system then creates the game with the users pieces being on the selected side

Post Conditions
● Game is started with user playing from their selected side

Includes and Extension Points
Included by: CreateNewGame

Special Requirements

Business Rules

Other Notes

Appendix E.7: Cancel New Game Use Case

Use Case Identification and History			
Use Case ID	CancelNewGame		
Use Case Name	Cancel New Game	**Version Number**	
End Objective	Cancel the process of creating a new game, and resume the current game if one is already active		
Created By		**On (date):**	
Last Update By		**On (date):**	
Approved By		**On (date):**	
User/Actor	Player		
Business Owner Name		**Contact Details**	
Trigger	User issues the command to cancel the game creation following the initial command to create a new game		
Frequency of Use	Can occur any time during the CreateNewGame process		

Preconditions
● User is in the CreateNewGameProcess

Basic Flow		
Step	**User Actions**	**System Actions**
1	User issues command to cancel the game	System will abort the CreateNewGame process

Exception Flow		
Step	**User Actions**	**System Actions**
1	The player issues a vocal command to cancel the game, when user is not in the CreateNewGame Process	The system audibly informs the player that they are not in the CreateNewGame process.

Post Conditions
● User is given the option to Create a New Game(Normal Fow)

Includes and Extension Points
Extends CreateNewGame

Special Requirements

Business Rules

Other Notes

Appendix E.8: Resign Game Use Case

Use Case Identification and History			
Use Case ID	ResignGame		
Use Case Name	Resign Game	**Version Number**	
End Objective	Concede defeat in the current active game and end the game		
Created By		**On (date):**	
Last Update By		**On (date):**	
Approved By		**On (date):**	
User/Actor	Player		
Business Owner Name		**Contact Details**	
Trigger	User issues command to resign the game		
Frequency of Use	This use case can be invoked once per game.		

Preconditions
● There is a game currently active. ● It is the user's turn.

Basic Flow		
Step	**User Actions**	**System Actions**
1	The user issues a command to resign the game	The system moves to the EndGame use case

Exception Flow		
Step	**User Actions**	**System Actions**
1	It is not the user's turn or there is not an active game	They system notifies the user of the error and no changes are made

Post Conditions
● The system moves to the EndGame use case

Includes and Extension Points
Includes: EndGame Extends: ReceiveVocalCommand

Special Requirements

Business Rules

Other Notes

Appendix E.9: Move Piece Use Case

Use Case Identification and History			
Use Case ID	MovePiece		
Use Case Name	Move Piece	**Version Number**	
End Objective	Move a piece in the currently active game		
Created By		**On (date):**	
Last Update By		**On (date):**	
Approved By		**On (date):**	
User/Actor	Player		
Business Owner Name		**Contact Details**	
Trigger	User issues command specifying a move sequence.		
Frequency of Use	This use case can be invoked an arbitrary number of times per game.		

Preconditions
● There is a game currently active. ● The specified move sequence is valid for current half-turn (i.e. the specified piece belongs to the player whose turn it is). ● The specified move sequence is not otherwise invalid (moving into check, failing to move out of check, or an invalid castle). ● The current half-turn does not belong to the chess engine.

Basic Flow		
Step	**User Actions**	**System Actions**
1	The user issues a command to move a piece.	The system validates the command against the appropriate preconditions and updates the current game board accordingly.

Exception Flow		
Step	**User Actions**	**System Actions**
1	The user issues an invalid command to move a piece.	The system notifies the user of the error and makes no changes to the board state.

Post Conditions
● The requested piece is moved appropriately (normal flow). ● The game state is as it was previously (exceptional flow). ● Will invoke EndGame if the move results in checkmate

Includes and Extension Points
Extends: ReceiveVocalCommand Extended by: EndGame

Special Requirements

Business Rules

Other Notes

Appendix E.10: Save Game Use Case

Use Case Identification and History			
Use Case ID	SaveGame		
Use Case Name	Save Game	**Version Number**	
End Objective	The program will the save the current game state so that the user can return to it at a later time		
Created By		**On (date):**	
Last Update By		**On (date):**	
Approved By		**On (date):**	
User/Actor	Player		
Business Owner Name		**Contact Details**	
Trigger	User is exiting a game and wants to save the game state.		
Frequency of Use	This use case can be invoked an arbitrary number of times per game, but will only be invoked if the user wants to exit the game and save the game state.		

Preconditions
● There is currently an active game to save. ● Player is exiting the game and wants to save the game

Basic Flow		
Step	**User Actions**	**System Actions**
1	From ExitGame use case	System asks the user if they would like to save
2	User selects “Yes”	System saves the game state, overwriting the previous save

Alternate Flow		
Step	**User Actions**	**System Actions**
1	From ExitGame use case	System asks the user if they would like to save
2	User selects “No”	System does not save the game state

Post Conditions
The game can be loaded on command

Includes and Extension Points
Extends: ExitGame

Special Requirements

Business Rules

Other Notes

Appendix E.11: Load Game Use Case

Use Case Identification and History			
Use Case ID	LoadGame		
Use Case Name	Load Game	**Version Number**	
End Objective	Load the saved game state		
Created By		**On (date):**	
Last Update By		**On (date):**	
Approved By		**On (date):**	
User/Actor	Player		
Business Owner Name		**Contact Details**	
Trigger	The user would like to load the saved game state		
Frequency of Use	This use case can be invoked any number of times as long as there is a saved game state		

Preconditions
There is a saved game state

Basic Flow		
Step	**User Actions**	**System Actions**
1	User issues command loading the saved game	System notifies the user that the current game will be ended and asks to continue
2	User selects “Yes”	System loads the saved game state and ends the current game

Alternate Flow		
Step	**User Actions**	**System Actions**
1	User issues command loading the saved game	System notifies the user that the current game will be ended and asks to continue
2	User selects “No”	System does not load game

Exception Flow		
Step	**User Actions**	**System Actions**
1	User asks to load game but there is no saved game	System notifies the user that there is no saved game

Post Conditions
Saved game state is loaded

Includes and Extension Points
Extends: ReceiveVocalCommand Includes: EndGame

Special Requirements

Business Rules

Other Notes