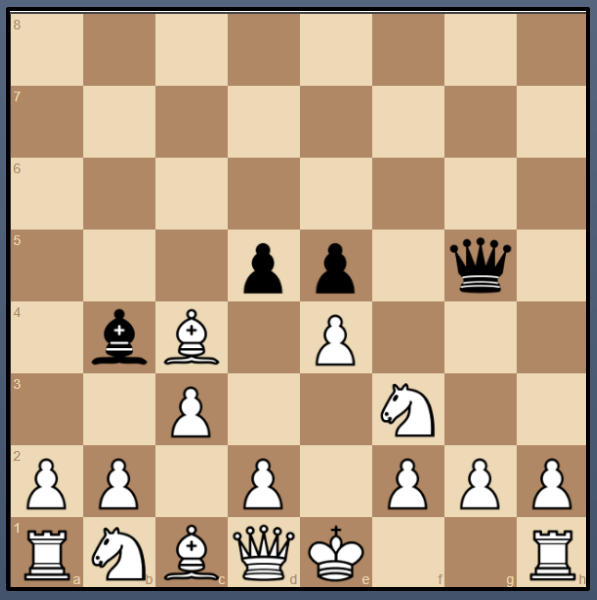
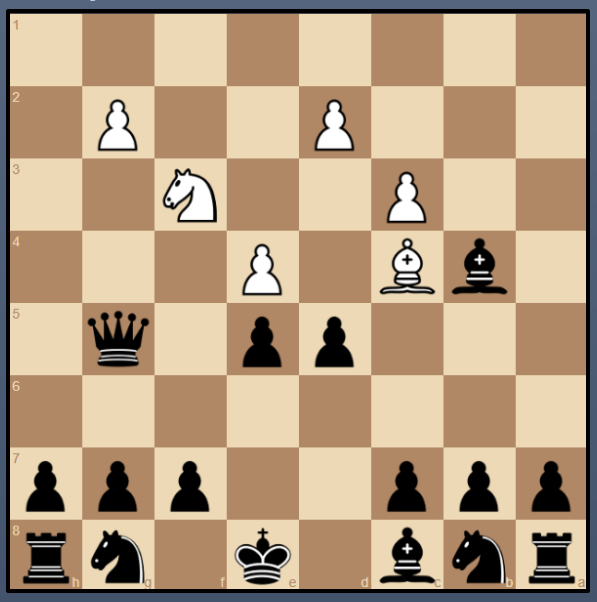
2019

DARPA CASCADE

Clausewitzian Chess: Companion Guide





FRICTION, FOG OF WAR CHESS VARIANT: DESIGN AND GUIDE

MAY 2019

This material is based upon work supported by the Defense Advanced Research Projects Agency under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency.

Table of Contents

Executive Summary

Purpose

This document serves as a companion to the Clausewitzian chess software tool. This tool is a chess variant designed to demonstrate the possibility of modeling complex cognitive concepts in simple, well-known games. More specifically, the effort that led to the tool's creation focused on understanding the complications that arise in complex systems of systems. It is important to note that in its current form the tool is an engineering prototype. We leave further polishing to future efforts and/or to the open source community.

We vary the standard rules and play of chess to directly model fog of war, friction, and asymmetric ends (limited versus total warfare). In this context, we define *fog of war* as hidden or uncertain information about one's self or one's adversary. Likewise, we define *friction* as the inability to receive or execute an order as desired. Chance heavily influences both of these notions. Finally, we have altered the notion of victory in the game of chess so that our variant is no longer a zero-sum game (both sides can now simultaneously win or lose). The Game Element Walkthrough section details these concepts.

This document describes all major game elements and includes a discussion of how each element manifests in the game and why it was included in the design. In addition, this document contains instructions on how to run the software tool and how to play the game, given the reader has access to the source code. Finally, the document discusses our findings from internal playtesting, known issues, and recommendations for next steps.

Findings

We have had multiple internal play testing efforts, both to find bugs and to gauge the types of gameplay that players experience. In total, our play testing incorporated more than 12 unique individuals who played collectively more than 25 games. While certainly not large, this sample size is enough for us to gain some insights into the game and its implementation. The primary findings from players were: 1) the fog of war component was reasonable and significantly changed how they played chess, 2) the friction component was "frustrating" and may even need to be relaxed in future versions, and 3) the asymmetric victory conditions and non-zero-sum outcomes successfully captured the dynamics of competing under conflicting ends, leading to different play styles and strategies for different combinations of victory conditions. Our conclusion is that we have successfully modeled fog of war, friction, and asymmetric ends within the game of chess. Additionally, we hypothesize that incorporating concepts into known games reduces some barriers for players to learn, exercise, or experience these concepts.

Known Issues / Next Steps

We categorize future efforts along three main lines. The first is fixing errors and problems that exist in the code base. The second is improving, adjusting, or balancing existing game elements. Finally, the third category is progressing on additional features that do not exist. As a general note, the tool would benefit significantly from polishing and user interface improvements; unfortunately, this was out of scope for the current effort.

Background

Program Overview

The Defense Advanced Research Projects Agency's (DARPA) Defense Sciences Office (DSO) has a number of research projects designed to advance the state of the art for the design, implementation, and management of complex systems. Specifically, the Complex Adaptive System Composition and Design Environment (CASCADE) program sought to "change how systems are designed for real-time resilient response to dynamic, unexpected contingencies." Furthermore, "The goal of CASCADE is to provide a unified view of system behavior, allowing understanding and exploitation of these complex interactions and a formal language for complex adaptive system composition and design. This unified view of system behavior, enabled by appropriate mathematical foundations, may also enable adaptation to unanticipated environments using arbitrary system components by providing a framework to dynamically identify and correct deficient system capabilities."[1]

To support this vision and provide a proving ground for advancements, we leveraged wargaming and other gaming approaches to investigate specific cases that test complex systems. Our earliest efforts often made simplifying assumptions, such as instantaneous and error-free communications, perfect order execution, and complete knowledge of the world state. In addition, the motivations of the adversary, when present, were typically worst-case obstructions of blue forces.

While these assumptions are all understandable and appropriate in context, we wanted to remove them from our game constructs. As such, we began modeling some of the real world complications that plague complex systems. Our initial efforts focused on domain and problem specific concepts, such as communication delays and coordination within specific command hierarchies. We found this was successful for specific games but not easily extensible to other problems and game structures.

The work described here is the outcome of our efforts to generalize our models. This work allowed us to capture the complications that arise in complex systems of systems and incorporate them into a simple game with which most people have some familiarity. We found that the Clausewitzian principles of fog, friction, chance, and total versus limited war[2] were well suited as a model framework. For the purposes of this report, we define fog of war as hidden or uncertain information about friendly and adversary units and friction as the inability to receive or execute an order as desired. For example, the effects of imperfect communication can be described in terms of the uncertainty (fog of war) created by the inability to communicate intention, position, or status as well as the inability to subsequently receive, and therefore perform, an order (friction).

Game Purpose

The purpose of the Clausewitzian chess variant is to show that it is possible to model complex concepts such as fog of war, friction, chance, and asymmetric ends simply in a known game structure. There are two implicit elements to this goal. The first is to show that we can model these concepts in a simple and generic way, which is helpful in understanding the concepts and demonstrating that it is possible to provide system analysis with these factors represented. The second is to show that we can build these models into existing games. Demonstrating this is important, as it shows that our models of fog, friction,

[1] https://www.darpa.mil/program/complex-adaptive-system-composition-and-design-environment
[2] Clausewitz, C., & Maude, F. N. (1982). On war. Penguin UK.

and asymmetric ends are flexible and compatible with new and existing games as well as modeling and simulation efforts.

To achieve these ends, we vary the standard rules and play of chess to directly model fog of war, friction, chance, and asymmetric ends (limited versus total warfare). In addition, we have altered the notion of victory so that our variant is no longer a zero-sum game (both sides can now simultaneously win or lose). The Game Element Walkthrough section details these concepts.

It is important to note that in its current form, the tool is an engineering prototype; therefore, one should expect to encounter some bugs and lack of beauty in the user interface. We leave further polishing to future efforts and/or the open source community.

Technology Transition

Deployment Options

In this section, we provide two sets of instructions in parallel. The first is for a Windows environment; for this platform, we provide a deeper level of detail and some additional help, such as scripts. Secondly, we provide some guidance on how to deploy on a Linux based machine in a cloud environment. We provide fewer details for this option.

These instructions highlight the two deployment options we are targeting. The first is a Windows based off-line environment designed for a security aware environment. In this setup, a Windows machine will run the server and connect to a router (wired or wireless). Client machines connected to the router can then navigate to the IP address of the server via a browser to play the game. The server machine stores all data. The second deployment option presented is a Linux cloud based environment such as a networked VM or Amazon Web Services. As this option is more advanced and specific to the actual deployment environment, we provide fewer details. For this option, we recommend consulting with someone who has the requisite skills to assist. For additional guidance or clarification, please contact the MIT LL team.

Installation / Dependencies

Install Project Files

The first step is to download the correct project files. If the source code is directly accessible as a zip file, unzip the files into the desired directory. If installing from Github, there are myriad options for downloading the project files, including downloading the source as a zip file or cloning the repository. For help with the latter, we suggest referring to the latest Git help documentation.

Dependencies

The second step is to install and configure all required dependencies. There are two primary required dependencies and a third optional dependency that is most useful for the cloud deployment option. In either case, it is helpful but optional for both deployment options.

The first dependency is the Node.js and npm environments (latest or LTS version). For Windows, one can download these and install using all default settings at the same time from the official Node.js website (https://nodejs.org/en/). To install in a Linux environment (e.g., Ubuntu), use the following commands in the terminal:

```
sudo apt install nodejs npm
```

Once complete, in both Windows and Linux, one can check the versions of each to verify installation. For Windows and Linux, one can run the following commands from a terminal.

```
node -v
npm -v
```

For Windows, we provide a script in the source code called “checknode.bat” which will show the versions of Node.js and npm if installed correctly. If installed correctly, double clicking on this script will pop up a command prompt window, show two different version numbers, and pause before closing.

Once this test passes, use npm to install the external dependencies for the project (requires internet connection). Running the following command in a terminal from the main directory containing the source code completes this step:

```
npm install
```

For Windows, we provide a script in the source code called “nodeinstall.bat” which, when double clicked, executes the command above. If the installation times out, the most likely cause is a proxy. If this is the case, please follow the instructions provided here to resolve and then retry running the script/command. To configure npm proxy settings, run the following commands from a command prompt / terminal:

```
npm config set proxy http://proxy.company.com:8080
npm config set https-proxy http://proxy.company.com:8080
```

The second required dependency is the database management system MongoDB. For Windows, one can download MongoDB from the following website: https://www.mongodb.com/download-center/community. Their website also offers detailed installation instructions: https://docs.mongodb.com/v3.2/tutorial/install-mongodb-on-windows/. Choose the complete setup and use all defaults. Note, this should automatically download and install the MongoDB Compass tool, which is covered in more detail in the Accessing the Data section of this report.

To install MongoDB on Linux (Ubuntu) run the following command.

```
sudo apt install mongodb
```

Once installed, the next step is to configure the database for the server. To configure manually on both Windows and Linux, run the following commands to create the database and add the appropriate user for the server.

```
mongo
use dcchess;
db.createUser({user:"mongouser", pwd:"mongopass", roles: [{role: "readWrite", db: "dcchess"}]});
show users;
```

For Windows, we provide a script in the source code called “dbconfig.bat” that provides this functionality. If the script completed successfully, it will show a message echoing the addition of the user. If run multiple times, the script will show an error saying that it was unable to add the user as the user already exists.

For further configuration testing and instructions on how to access the data stored within, refer to the Accessing the Data section of this report.

The final and optimal dependency is a Node.js process manager named pm2. pm2 starts, stops, and manages the server while it is running. It provides monitoring and handling to handle cases where the server may shut down. For more details, see their website: http://pm2.keymetrics.io/. While optional, we strongly recommend the use of pm2 for the cloud deployment option. It is helpful in the Windows environment but not necessary.

In both Windows and Linux, one can install pm2 using the following command:

```
npm install pm2 -g
```

For Windows, we include a script that will handle the install called "pm2install.bat." pm2 usage and help scripts will be covered in the Running the Server section of this report.

Cleanup

To ensure proper installation and caching, run the following command after installing and configuring all dependencies.

```
npm install
```

For Windows, running the "npminstall.bat" script completes this action.

Running the Server

The general workflow for starting the server is to ensure the database is running and then launching the web application. In this section, we will show how to manually execute these steps as well as provide an overview of the help scripts for the Windows environment.

Manual

The first step is to ensure the MongoDB client is running. There are many ways to start the database. The MongoDB documentation includes full instructions. At a minimum, one can ensure the database is started (provided the default installation directions were followed) by starting the mongo.exe in Windows located by default in "C:\Program Files\MongoDB\Server\4.0\bin".

Once the database is running, one can start the web application server in one of two ways. The first is without pm2. To achieve this, execute the following command in a terminal in the project's main directory.

```
node app.js
```

The second is with pm2, as shown below. To achieve this, execute the following command in a terminal in the project's main directory.

```
pm2 start app.js
```

We recommend using pm2 as it provides extra monitoring and handling of the web application. Additional pm2 commands for listing, monitoring, starting, stopping, restarting, and inspecting the web application are provided below in respective order. When "0" is used, it is referring to the application's id in the pm2 reference list.

```
pm2 list
```

```
pm2 monit
pm2 start app.js
pm2 stop 0
pm2 restart 0
pm2 show 0
```

Scripted

For the Windows deployment option, we provide some scripts to automate this process. These are as follows:

- startdb.bat -> makes sure the MongoDB database is running
- startapp_node.bat -> starts the server without pm2
- startapp_pm2.bat -> starts the server with pm2
- restartapp_pm2.bat -> restarts the server with pm2
- monitorapp_pm2.bat -> monitors the server with pm2
- stopapp_pm2.bat -> stops the server with pm2

The recommended use case when not using pm2 is to first run startdb.bat followed by startapp_node.bat. This will result in two separate command prompts / terminals being open while the server is running. Similarly, the recommended use case when using pm2 is to first run startdb.bat followed by startapp_pm2.bat.

Running the Game

When the server is successfully up and running, clients are able to connect to the server using port 3000. To run the game on the server machine, open a web browser, and type the following into the address bar “localhost:3000”. This will load the web application if the server is successfully running. Note, it is possible to connect more than once from the same machine using different browser windows or tabs. Using this approach, it is possible to play the game completely from a single machine fully offline.

Other machines can also serve as clients and run the game, provided they can make a connection to the server machine. If the server and client machines are all connected to a router (no internet connect required), the clients can make a connection by opening a browser and typing the following into the address bar “[server IP address]:3000” where [server IP address] must be replaced with the IP address of the server. One can detect their local IP address by running the “ipconfig” command from the Window’s command prompt or “ifconfig” on a Linux machine. For offline/local play, the IP address will most likely resemble “192.168.X.X”.

We do not provide advanced instructions for configuration in a cloud environment. However, common steps are configuring the cloud to run the server, obtain a domain name, and configuring the system to link webserver to the domain name. There are numerous resources online to aid in this deployment option.

Trouble Shooting

If there are issues, the best course of action is usually to restart the server. If using pm2, one can restart the server by double clicking on the restartapp_pm2.bat or executing the following command:

```
pm2 restart 0
```

If not using pm2, close all terminal windows. This will shut down the server started using the startapp_node.bat script. One can then start the server again by re-running the startapp_node.bat script or the following command:

```
node app.js
```

In addition, it is often helpful to have clients close their browser and reconnect. This solves most client side issues without requiring a server restart.

Accessing the Data

The database used for this application is MongoDB, which has a different storage and query style than traditional relational database management systems, such as MySQL or Microsoft Access. As such, standard SQL queries are not used. For an intro into the use of MongoDB, we suggest the following tutorial: https://docs.mongodb.com/manual/tutorial/.

To make queries against the database manually, first start the database and then execute commands in the prompt. For Windows users, we provide a script to start the database named "startdb.bat" that will load the MongoDB interface for querying. This can be done manually by starting mongo or running the mongo.exe executable from "C:\Program Files\MongoDB\Server\4.0\bin".

It is possible to access and query the data through a user interface, which might reduce the complexity. This interface tool is MongoDB Compass. The tool is available for download here https://www.mongodb.com/download-center/compass. For full installation instructions, refer to the MongoDB documentation here: https://docs.mongodb.com/compass/master/install/.

For additional help with custom queries or analytics contact the MIT LL Team.

Shutting Down

For the offline/local Windows deployment option, one should shut down the server when finished. If the server was started without pm2, this can be achieved by pressing "Ctrl+C" in the command prompt or terminal running the server. Alternatively, one can close this terminal severing the connection. If using pm2, one can stop the server by running the "stopapp_pm2.bat" script or the following command:

```
pm2 stop 0
```

For the cloud-based deployment, there is no need to stop the server.

Game Elements and Design

Basic Principles

This section will detail each game element. We will highlight where each element manifests itself in the tool, provide a brief description of how it works, and discuss our design rationale. In general, we will discuss game elements in terms of how they differ from the ones employed in a traditional chess game. We assume the reader has at least an introductory knowledge of chess. If this is not the case, we strongly recommended that the reader do some research to understand the basics of chess, including the pieces, how pieces move, the traditional layout, capture, combat, win conditions, and introductory tactics. LiChess provides an excellent free tutorial (at the time of publication) at this location: https://lichess.org/learn#/.

The flow of Clausewitzian chess is simple. First, log in and join a game. Second, setup the pieces within a budget (we refer to this mechanic as *piece composition*). Third, gameplay (chess with *fog* and *friction* present). Fourth, determine a winner (chess with *asymmetric ends* present).

Core Game Mechanics

There are four major mechanics that diverge from traditional chess. The first is in how the game begins. In traditional chess the board starts with a fixed layout; the player does not have the ability to modify their starting position. Some chess variants begin with randomized positions, but Clausewitzian chess allows for full player customization within a fixed budget (piece composition). Second is the inclusion of friction. In traditional chess, players always have full control over their pieces and can move them perfectly as long as the move is valid. In Clausewitzian chess, players sometimes have no control over some of their pieces (friction). The third is the inclusion of fog of war. In traditional chess, players always have full visibility of the board, including their own and their opponent's pieces. In Clausewitzian chess they do not have full visibility of their opponent's pieces (fog of war). Finally, traditional chess ends with King Capture. There is no maximum number of turns and the game is zero-sum, that is, if one player wins, the other must have lost (it is also possible to end in a draw). In Clausewitzian chess, there are multiple victory conditions and the game is no longer zero-sum. This means that both players can win and both players can lose (there are no draws allowed).

All game elements that differ from traditional chess are a result of one of these departures. These departures, however, result in the emergence of interesting gameplay.

Game Element Walkthrough

Login System

The login system is the first thing the user sees. The portrait shown is of Carl Von Clausewitz, the person after which we named the game. The login system does not provide authentication; it simply associates a connection to the server with the username provided allowing the player to be identified in the game and chat. We leave any authentication into the system to the reader. The figure below shows the login feature.

Figure 1: Login System

Lobby System

Upon logging in, the player will see the lobby system. They will see their username and a "Logout" button, which will bring them back to the login system. In addition, they can see a list of active games and a list of players who are online and available for play. To begin play with a player, click on their name.

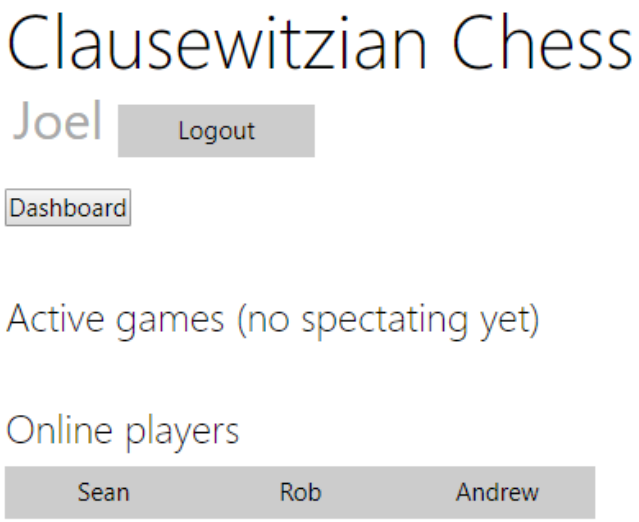


Figure 2: Lobby System

Players

The game is currently a two-player, human versus human game. Therefore, one must pair up with another user to launch a game. Note, it is possible to play with a single player, but in order to do so, one must open two different browser tabs or windows, log in twice using two different names, and launch a game. To reiterate, to begin a game with another player, click on their name.

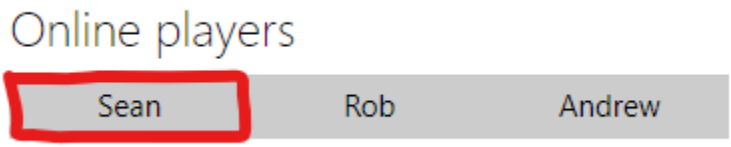


Figure 3: Player Selection

Game Invites

Once a player has initiated a game with another player, both players will receive a pop-up. The player who sent the invite will see the following:

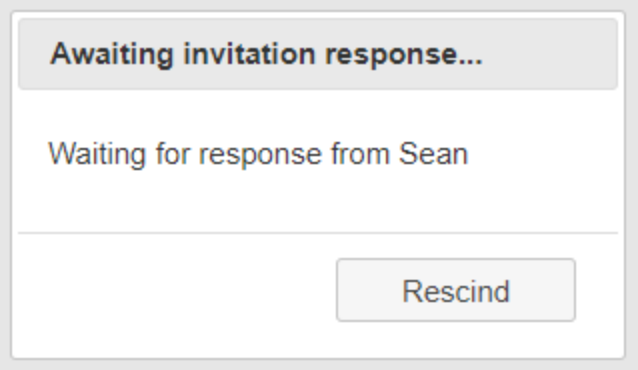


Figure 4: Sent Invite

The player who received the invite will see the following:

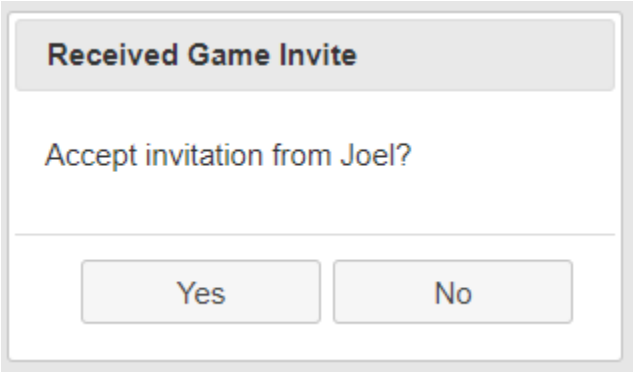


Figure 5: Received Invite

The reason we implemented an invite system instead of automatically joining was to allow users more control over selection of their opponent. In addition, rescinding invites allows for better error handling and correction of accidental invitations.

Flow Control

Once in the game, a player has the ability to exit in one of two ways. The first is Logout (which is also available from the lobby system). The second is the Resign button. Both result in losing the game. In addition, if a player closes the browser tab or window, this will have the same effect as resigning and then logging out. We recommended players Resign and/or Logout instead of directly closing the browser tab or window. The figure below shows these controls.

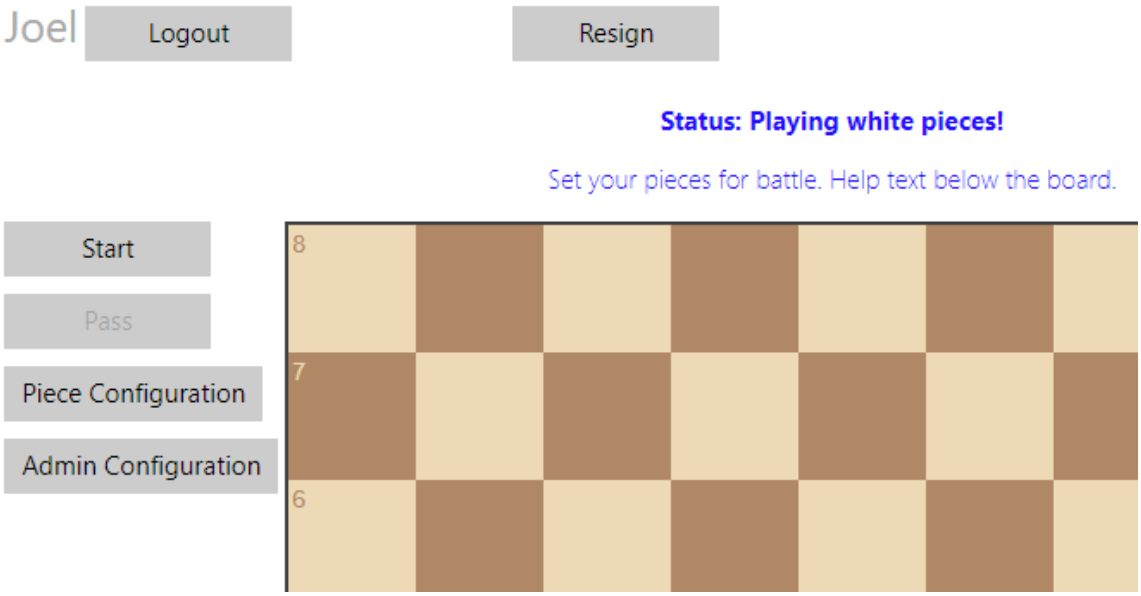


Figure 6: Logout and Resign Buttons

Currently there is no way to re-join a game once a player has exited. This may be included in future iterations.

Chat System

The chat system provides a means of communication between players. Currently the game setup assumes some level of coordination and trust between players. Players should discuss and agree on both piece configurations and admin configurations before making changes. Players may do this outside of the game or have someone provided configuration instructions to them (as in a classroom setting). However, should players be separated by distance, the chat feature allows them to settle on the rules of the game together.

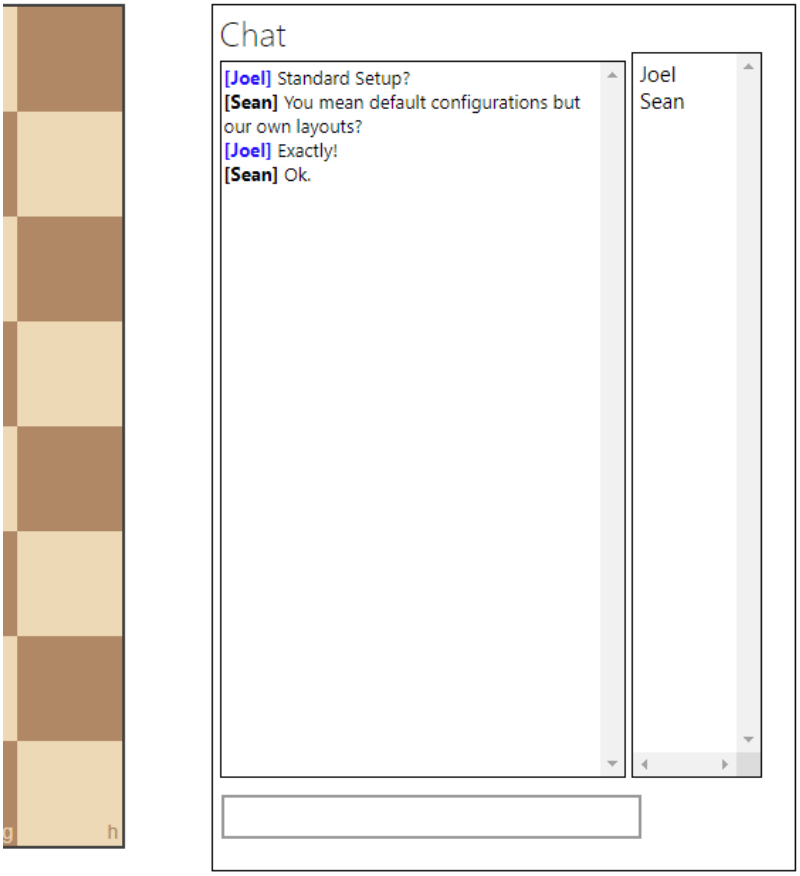


Figure 7: Chat Area

Game Setup: Piece Configuration

There are myriad possible configurations for gameplay. The game options associated with the values and operations of pieces is located in the Piece Configuration Panel. Clicking on the "Piece Configuration" button shown below activates the panel.

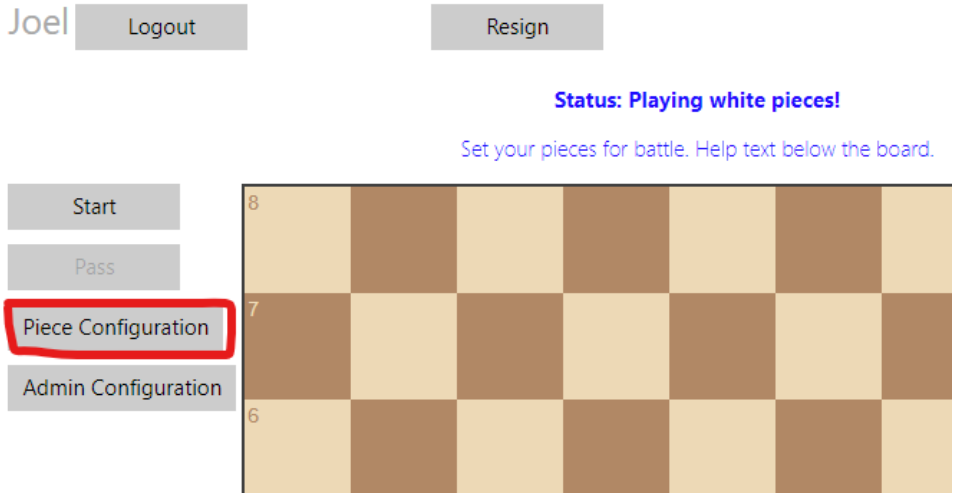


Figure 8: Piece Configuration Button

Within this panel, one can change the attributes of pieces for both sides as shown below.

Figure 9: Piece Configuration Panel

Each row controls a class of pieces. For example, “White Pawns” sets the values for all white pawns, not just a particular one. Similarly, “Black Rook” changes the values for all black rooks on the board. The last row sets values for the entire column (all pieces of both colors).

Each column represents a different property. The “View Range” property identifies how many squares the specified pieces can see in terms of distance. This includes all directions including diagonals. The “Sight Strength” property identifies the level of resistant to jamming for the specified pieces. The “Jamming” property identifies the jamming ability for the specified pieces. An opponent’s piece is visible on the board if 1) the piece is within the “View Range” of a friendly piece, 2) if the “Sight Strength” of the “in-range” friendly piece is greater than the “Jamming” value of the opponent’s piece. In addition, the “See Inward Threats” checkbox makes visible all pieces that are attacking friendly pieces. Similarly,

the “See Outward Threats” checkbox makes visible all pieces that friendly pieces are attacking. The “Friction” checkbox signifies if pieces are vulnerable to friction. The final attribute, “Value” is how much material the pieces are worth. That is, if a player captures a piece with value of three, this counts as capturing three material.

It is important to note that the game synchronizes changes made between both players in the game. In addition, the game updates the configuration panels immediately. If a player makes changes, the other player can easily see the changes by opening the Piece Configuration panel.

It is also important to note that either player can change settings for both players at any point in the game, even after gameplay has started. We chose this approach to configuration to allow maximum flexibility under a use case in which players are approaching the game in good faith. For some online environments or those where there is an incentive to cheat, one may desire to implement additional controls to ensure a fair and equitable handling of the configurations.

Game Setup: Admin Configuration

The Admin Configuration game element is very similar to the Piece Configuration element except it contains game options that deal with general gameplay, not pieces. The same caveats and reasoning apply in terms of who can make updates and why there are so few controls restricting the ability to edit settings. To open the admin panel, click on the “Admin Configuration” button as shown below.

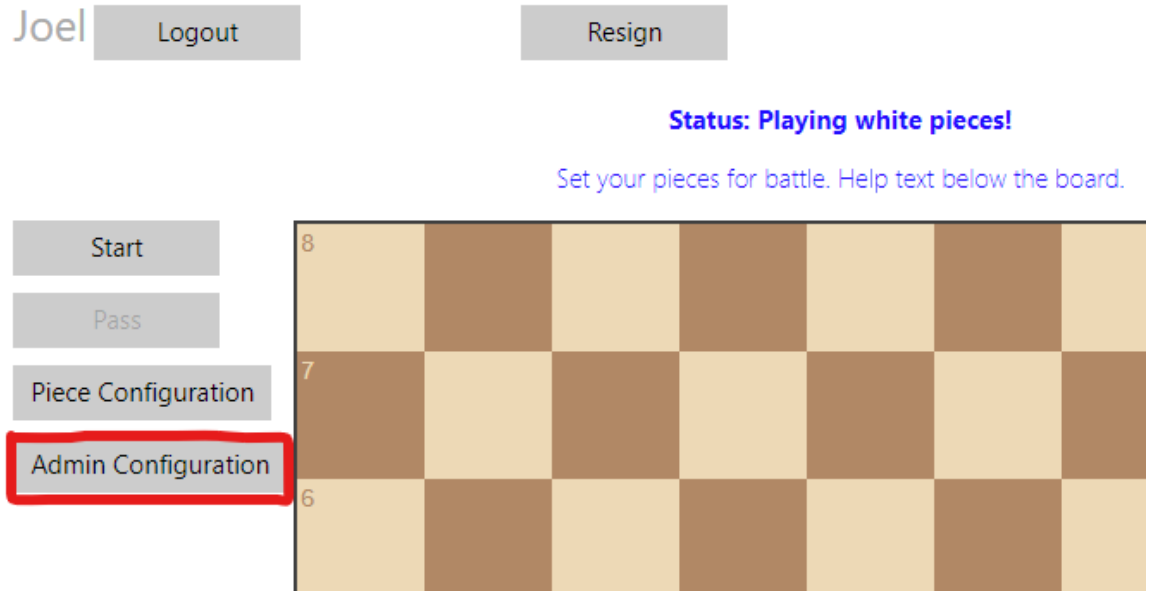


Figure 10: Admin Configuration Button

The figure below shows the game options available.

Figure 11: Admin Configuration Panel

The first option sets the "Maximum Turns" in the game. The second sets the "Maximum Budget", which is used in the Piece Composition game mechanic. The "Victory Condition" option allows a user to overwrite their own victory condition. Note, this is the only option that the game does not synchronize with the opponent; that is, changing the victory condition using the Admin Configuration will only update a player's victory condition. The remaining check boxes control the information shown below the game board during gameplay. "Show Scores" displays the current scores for the different victory conditions. "Show Current Friction" shows the current friction piece. The "Show Next Friction" shows the friction pieces for the next turn. "Show Captured Pieces" shows a running list of the pieces captured. Finally, "Show Adversary Scores" shows the opponent's scores for each victory condition.

Piece Composition

The Piece Composition game mechanic allows a player to set their own custom starting position. We included this feature to enrich the player's choices and capture the notion of "composition" as used in the CASCADE program. In short, players can compose their forces (composition), control their execution (orchestration), and adapt their plans on the fly (limited adaptation; not able to re-compose pieces mid-game).

After joining a game, the game places the player directly into the Piece Composition view as shown below. Note how there is a row of spare pieces along the bottom.

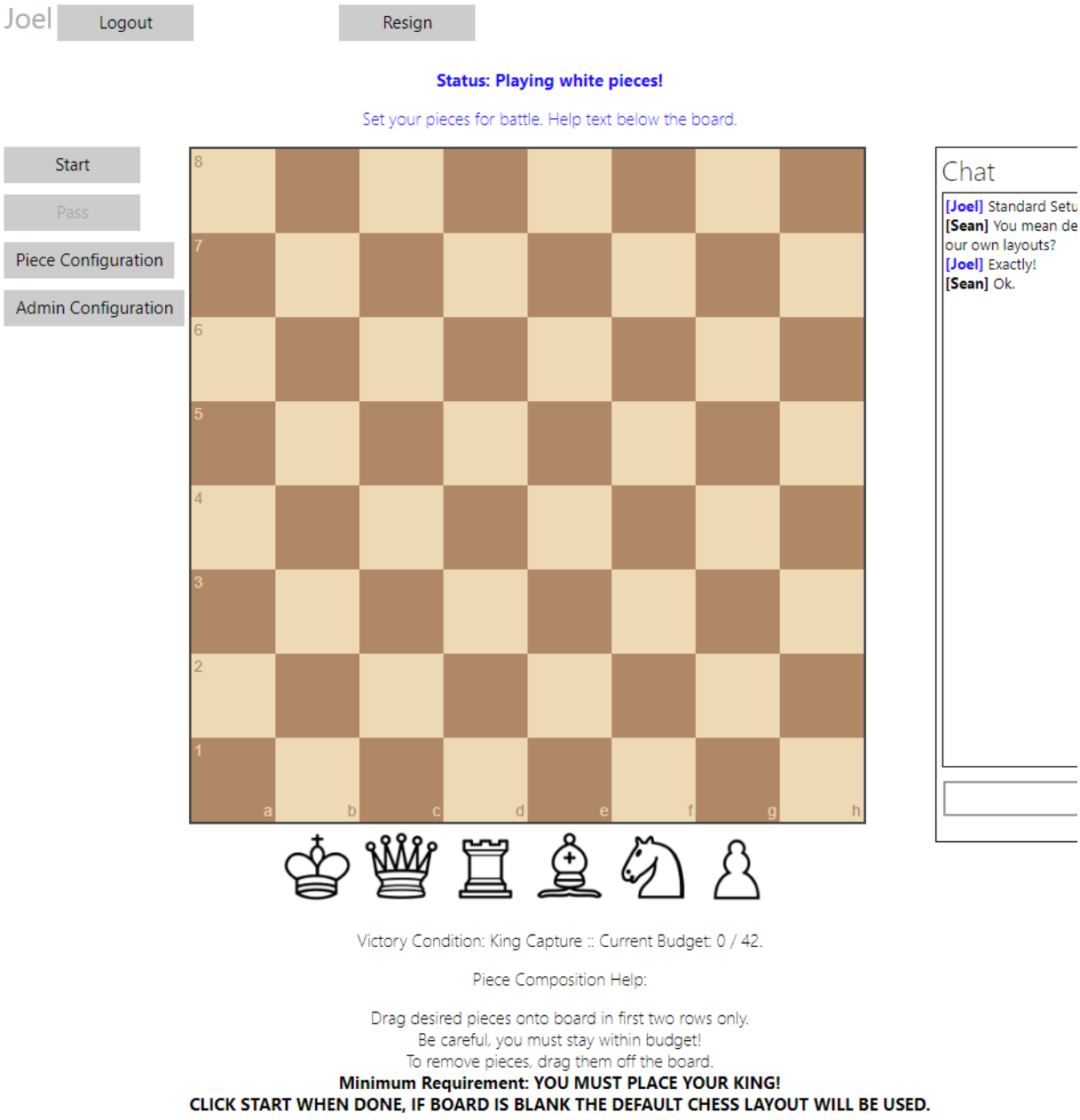


Figure 12: Piece Composition (Setup)

There are a few key things to highlight in this figure. First, the Status above the board the player's piece color. Second, there is some help text below the board, which provides the basic instructions for placing pieces. Third, within help text below the board shows the player's current Victory Condition and the current budget. Knowing the victory condition will most likely change how a player deploys their pieces.

The basic controls of Piece Composition are dragging and dropping pieces. To add pieces to the board, drag them from the spare pieces below the board onto the board in the desired location. Once on the board, the player can drag pieces to alternate locations. To remove a piece, drag it off the board. The rules for piece composition are simple. Each piece has a value, which is set in the Piece Configuration panel. When a player places a piece, the piece's value counts against the total budget, which is set in the

Admin Configuration panel. Players can only place pieces in the bottom two ranks; if a player tries to place a piece elsewhere on the board, the piece will snap back to its original position. If placing a piece would exceed the total budget, one cannot drag the piece from the spare piece location. The only requirements for Piece Composition are that each player must remain under budget, players can only place pieces in the back two rows, and that each player must place their King. Finally, if a player would like to play with the standard chess layout, one can simply begin by clicking the "Start" button, shown below.

Once satisfied with the piece composition, click the "Start" button shown below to begin the game.

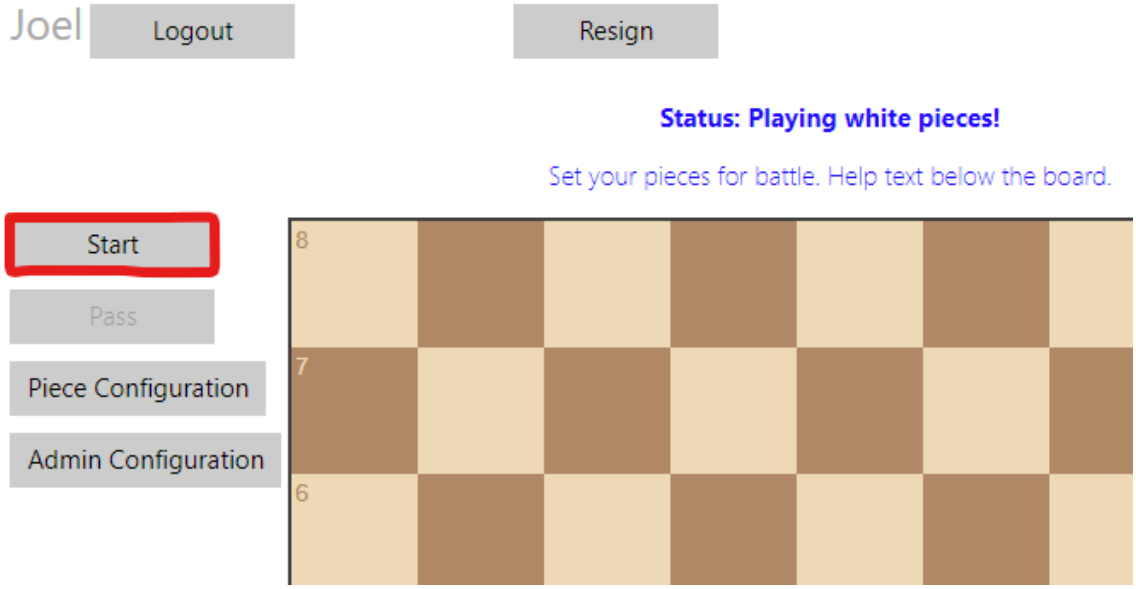


Figure 13: Start Button

Victory Conditions

In the current version of the game there are four victory conditions, summarized in the following table.

Table 1: Victory Conditions

Condition	Definition
King Capture	To achieve victory one must actually capture the other player's King. Unlike in normal chess where one would move into checkmate to end the game, this game does not observe the normal rules of check and checkmate. That is, the game does not notify players if they are in check and they are not required to move out of check. Capture the opponent's King just like any other pieces to win. Hardest to achieve of the four existing victory conditions.
Most Pieces	To achieve victory, one must have captured the greatest number of pieces (regardless of value/material) by the end of the game. Moderate difficulty.
Most Material	To achieve victory one must have captured the most material (regardless of the number of pieces captured) by the end of the game. Moderate difficulty.

Condition	Definition
Three Check	To achieve victory one must place the other player's King in check three times before the game ends. Note that this game does not observe the normal rules for check, that is, the game does not notify players if they are in check and they are not required to move out of check. While able, a player should NOT capture the opponent's King under this victory condition until they have checked it three times, otherwise, it is impossible to win. Easiest to achieve.

Each player begins the game with a random victory condition. This implies that each player has a good chance of having a different victory condition than their opponent. As such, the game is no longer zero-sum. That is, both players can simultaneously win or lose. For example, if player A has "Three Check" and player B has "Most Pieces," both can lose if player A fails to check player B three times and if Player A captures more pieces than player B. Similarly, if player A checks player B four times and Player A captures the most pieces, both players would win. Note that most of the victory conditions are in terms of the "end of the game," see the Number of Turns section for more details.

There is a special case when both players randomly select "King Capture." When this happens, the game goes into a special mode in which the maximum number of turns is set to infinity. Play will continue until one of the players physically captures the other player's King. When this happens, the game will automatically end. In all other cases, play continues when a player captures a King until the game reaches the maximum number of turns.

We chose these victory conditions to provide interesting combinations for games while allowing for asymmetric goals. In addition, the victory conditions provide an acceptable initial game balance. "Three Check" is probably the easiest of the victory conditions to achieve, but it is also the easiest to detect. The other victory conditions are more difficult to achieve but are harder to detect, reducing the ability of an opponent to frustrate a player's path to victory. In addition, having "King Capture" present models how different actors can have different goals of different difficulty in conflicts. These victory conditions, which are a small subset of what is possible, seemed the most balanced, interesting in combination, and exposed the player to dilemmas we desired to achieve.

Number of Turns

Unlike traditional chess, Clausewitzian chess has a maximum number of turns (with the exception of dual King Captures as outlined in the Victory Conditions section of this report). The maximum number of turns is defaulted to 20, but players can adjust this value in the Admin Configuration panel. A turn consists of a move by both players. Since white always begins, black always has the last move.

This system was necessary to break the zero-sum nature of chess and allow for more expressive and interesting asymmetric goals between players. In addition, a limit on turns induces a desirable pressure on players so they are constantly working toward a victory condition, which motivates interesting conflict on the board. Without this pressure, both sides could sit back and wait for the other side to make a mistake or reveal information.

Friction Pieces

The inclusion of friction pieces is one of the main game elements in Clausewitzian chess. Each turn a class of pieces is unavailable for the player to move. In some configurations (including the default), the

game also shows the friction piece for the following move, allowing the player to make plans for their future turns. In the example shown below, the player is unable to move their Pawns this turn and next turn will be unable to move their King.

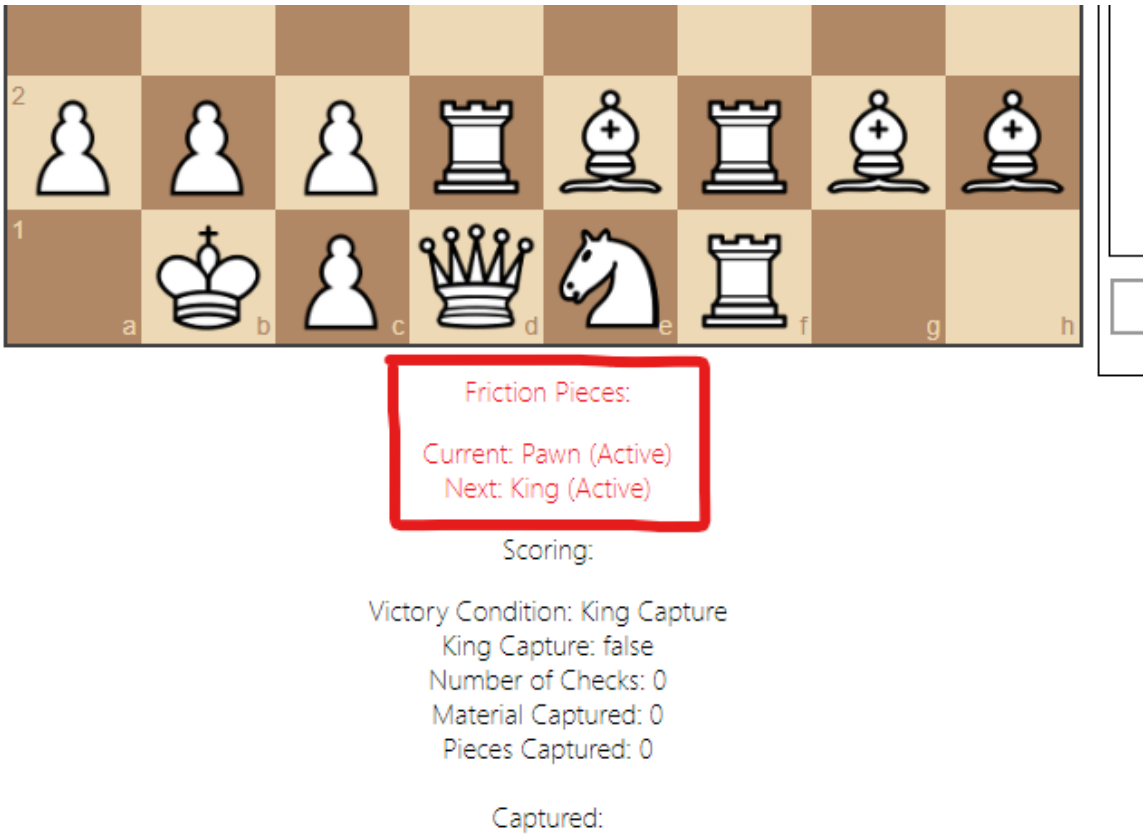


Figure 14: Friction Pieces

It is possible to hide all friction piece hints as well as to show any combination of the friction pieces for the current and next turn in the Admin Configuration panel. In addition, players can configure the game using the Piece Configuration panel so that some pieces are immune to friction. The friction status will show "Active" if the pieces shown are vulnerable to friction and "Immune" if the pieces shown are immune.

There are a few important things to note with regard to friction pieces. The first is that if a player attempts to move a piece experiencing friction, the game will allow the player to select the piece and drag it as if moving it. However, when a player attempts to place the piece at the new location, it will snap back and the player will see an updated status stating they cannot move friction pieces. The Game Status section of this report details and shows this status. The second thing to note is that the game selects friction pieces at random without constraint. The result is that a player may have the same friction piece for multiple turns in a row. In addition, the player may experience friction on a set of pieces that they do not have on the board. Finally, as a point of clarification, friction applies to all pieces of that type for a player. Therefore, if a player has three rooks on the board and "Rook" comes up as the friction piece, they will be unable to move any of the rooks if their rooks are vulnerable to friction.

We implemented friction pieces this way for specific reasons. Firstly, this method roughly captures real world friction. We define friction as the inability to receive or execute an order as desired. In the initial implementation, we model friction by prohibiting the execution of orders on pieces. A real world equivalent could be lost communications for which a desired unit does not receive an order. In future iterations of the game, we would like to expand piece friction so that there is a chance for friction pieces to execute the move desired with a delay and/or execute a slightly altered move. The second reason for implementing friction pieces this way is to minimize the indication of which pieces are under friction. In fact, the ideal setup may be with no indicators of friction present at all (although this may degrade the player's experience to an unacceptable level). This is important because we want players to consider all possible moves without providing an obvious framing. For example, if we highlighted the friction pieces on the board, players would obviously ignore considering those moves. The player fully experiences friction as intended when they have decided that a particular course of action is best and then are unable to execute against it. Instead, the players must adapt their plan and select a different move to work around the constraint. We believe our implementation of friction achieves this effect.

Turn Indicator

Once both players have chosen their force composition by setting their initial board layout, the game will start. As with traditional chess, the white player always moves first. The game status message above the board indicates the player's color.

To help players understand when it is possible for them to select a move, we have implemented simple turn indicators. These are to the left of the board under to play control buttons. When it becomes the player's turn, their color's turn indicator will flash at them for a few seconds. This is to help direct the player's attention back to the game since they were previously waiting on the opponent to make a move. The figures below show the white and black turn indicators respectively.

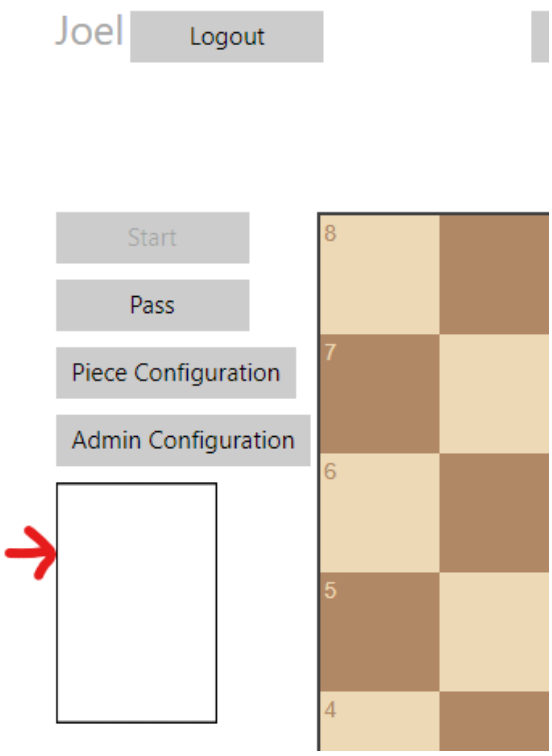


Figure 15: White Turn Indicator

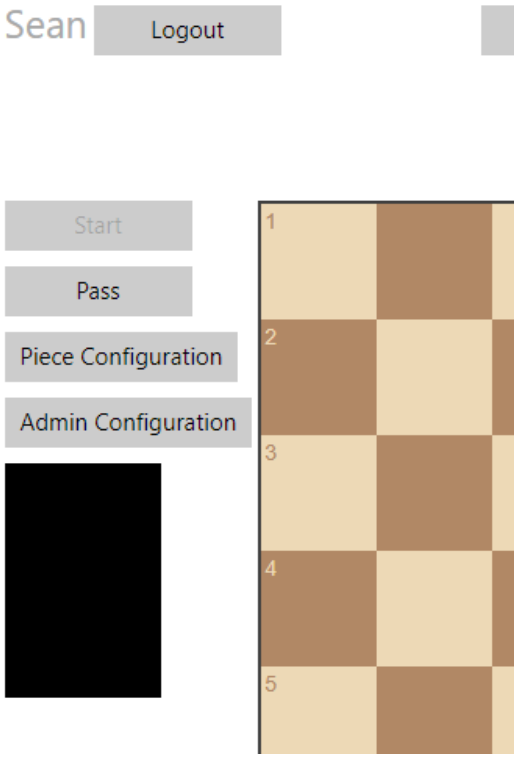


Figure 16: Black Turn Indicator

Chess Board and Gameplay

Once the game begins, gameplay progresses like traditional chess with players exchanging moves. There are a few notable exceptions. First, traditional chess has special moves for pawns and castling. The Special Rules section of this report discusses how Clausewitzian chess handles these special rules. Second, due to fog of war, pieces may unknowingly collide. This can happen if a player tries to move a piece like a Rook or Bishop multiple squares and there is a hidden piece that could block movement. In this case, if a player's move encounters a blocking piece, the moved piece will "bounce back" one square and land in the square in front of the blocking piece. In the special case that the move ends on a square with hidden piece with no blocking pieces in between, the player captures the hidden piece even though the piece was not visible. Third, gameplay continues to the maximum number of turns, as described in the Number of Turns section. This means that play continues after King Capture, there are no stalemates, checkmates, or other halts to gameplay. In many ways, this actually simplifies the rules of chess. Fourth, it may be necessary for a player to pass on their turn if no moves are available. This can happen by friction and in some other rare cases. It is important to note that a player will only be able to pass on their turn and that this action constitutes their full turn. We recommend players avoid passing unless necessary as players have limited tempo with which to achieve their victory conditions. The figure below shows the pass button. Note that the button is enabled (able to be clicked) as it is white's turn.

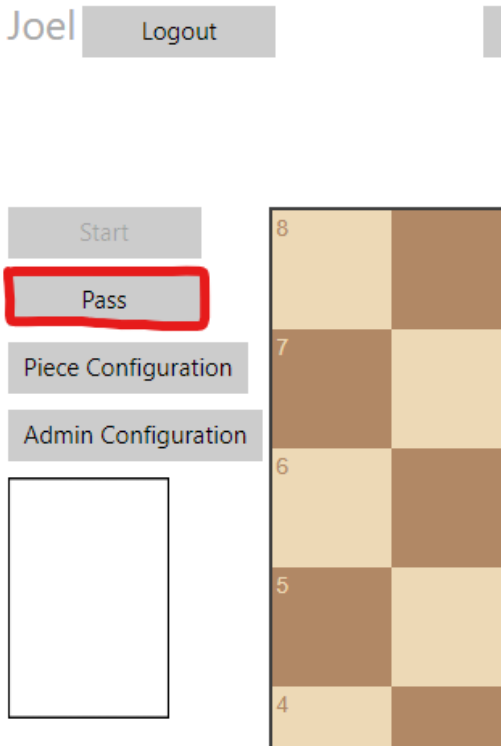


Figure 17: Pass Action

The flexibility provided by the Piece Configuration and Admin Configuration settings is important. Using different configurations, it is possible to play a game very similar to traditional chess. If the players set the "View Range" to eight and "Jamming" to zero, this will show the full board to both players throughout the game. This effectively "turns off" fog. Similarly, players can set "Friction" to unchecked for all pieces allowing the player to move all pieces even if they are friction pieces. This effectively "turns

off" friction. Lastly, if players both override their victory conditions to "King Capture" the players will be competing for the same symmetric victory condition as traditional chess. With all of these settings taken together, the only different between Clausewitzian chess and traditional chess is the ability to castle, check enforcement, literal King capture in lieu of checkmate, and no special outcomes like draw and stalemate. The flexibility of configuration also allows us to set up interesting experiments and control many variables. For example, it may be worth running an experiment in which fog and friction are both "turned off" to explore just the impacts of asymmetric victory conditions without confounders.

In addition to the flexibility provided by the configurations, different settings have drastic impacts on gameplay. Optimal play is extremely less obvious compared to traditional chess. In many cases, the best moves for traditional chess given the same board setup are terrible moves in Clausewitzian chess. This is partly due to fog and friction but mostly a result of differing and asymmetric victory conditions. The figure below shows the game board after the first turn; note that the turn indicator shows that it is black's turn to move.

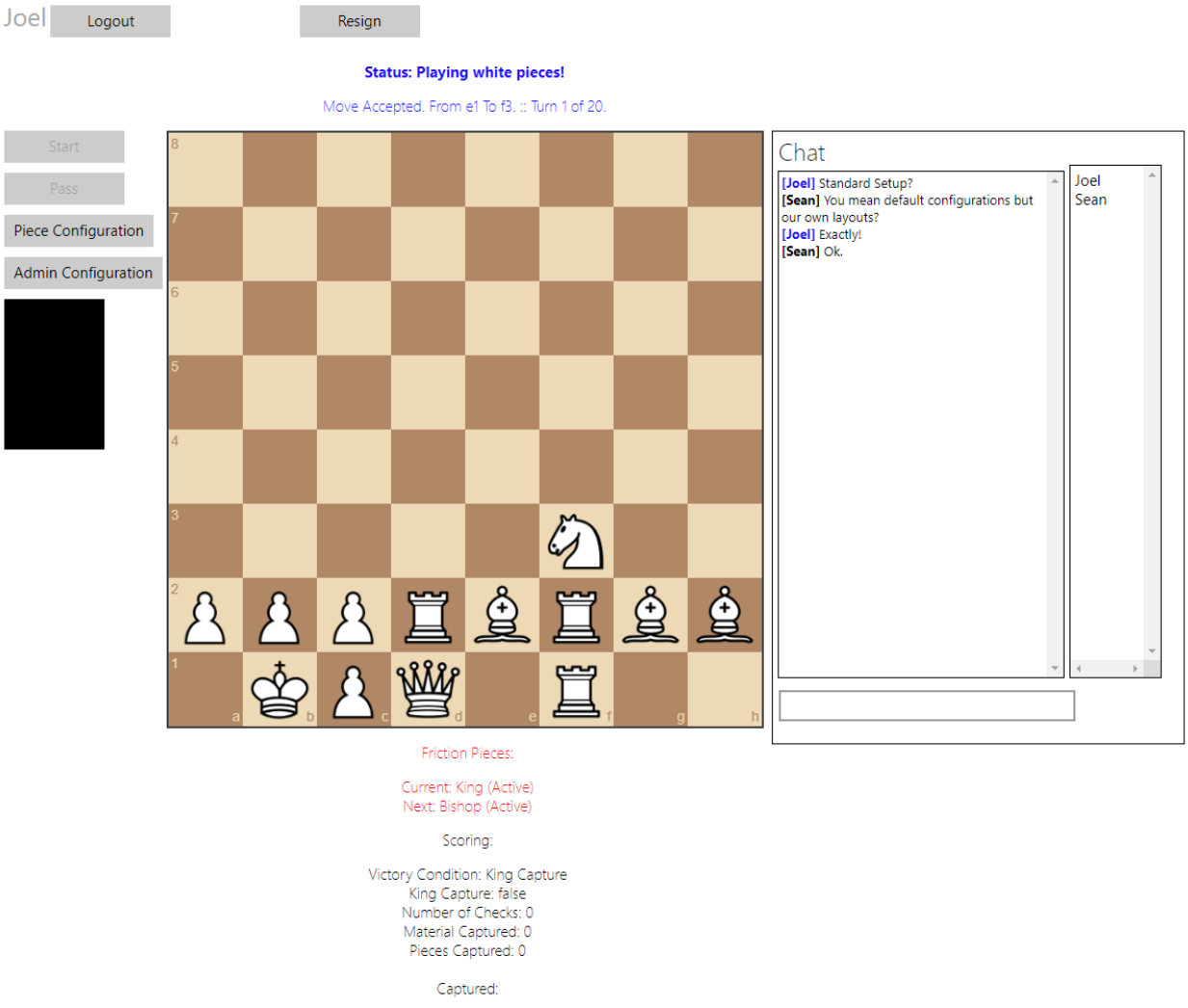


Figure 18: Gameplay

Game Status: Turns and Gameplay

During gameplay, the game will provide the player with basic status messages. Most commonly this will include feedback for successful moves, changes in configuration, or when a player attempted to make an illegal move. Most status messages should include the current turn and the maximum number of turns.

The text above the game board shows the general status of the game. As shown below, this includes a bolded header that tells the player what color they are playing followed by a line of text below it with a status message in blue text.

Figure 19: Status - Turns

Game Status: Friction Pieces

The text below the game board shows specific status updates for gameplay. The red text directly below the game board shows the status message of friction pieces.

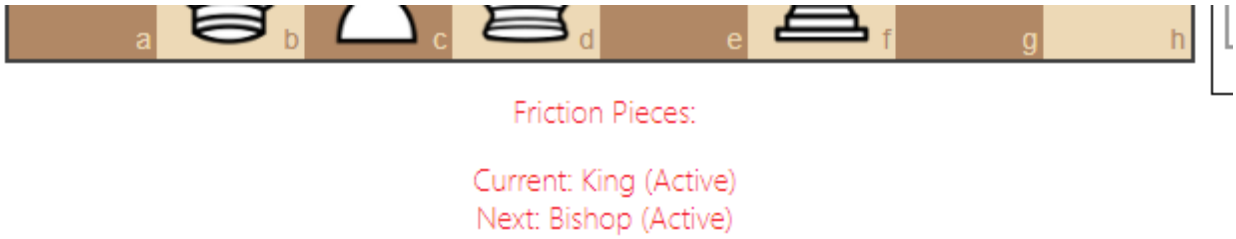


Figure 20: Status - Friction Pieces

Here we note that the status shows both the current and next friction pieces as well as if they are vulnerable to friction ("Active") or immune ("Immune"). See the Friction Pieces section for a full discussion on their role in the game. Note that in some configurations, the current or next (sometimes both) friction pieces may not be shown here. If this is the case, the red "Friction Pieces:" text will be visible but the red text below it indicating the friction moves will not be present.

Game Status: Scoring

The black text directly below the Friction Pieces status shows the he current scoring. At a minimum, the status includes the player's victory condition. Some configurations (settable in the Admin Configuration panel) can change the type and amount of information provided in this status section. The figure below shows the default setting. In this case, the status only shows the player's score information. Note that the status shows progress toward all victory conditions regardless of the player's actual victory condition.

Friction Pieces:

Current: Queen (Active)
Next: Bishop (Active)

Scoring:

Victory Condition: King Capture
King Capture: false
Number of Checks: 0
Material Captured: 19
Pieces Captured: 5

Captured:
Pawn, Pawn, Rook, Bishop, Queen

Figure 21: Status - Scoring (Standard)

In some cases it may be useful to show the adversary's score status as well. We include this option since this information is knowable to both players; tracking and showing the status is simply a memory aid. Technically, tracking and showing the score is not necessary as players can observe and deduce all captures and checks. Therefore, showing the player's and adversary's score status does not reveal any hidden information. Note, showing the adversary's score status does not reveal their victory condition; this always remains hidden. The figure below shows the score status that includes the adversary's scores, shown in brackets.

Friction Pieces:

Current: Queen (Active)
Next: Bishop (Active)

Scoring:

Victory Condition: King Capture [Adversary's Score]
King Capture: false [false]
Number of Checks: 0 [0]
Material Captured: 19 [3]
Pieces Captured: 5 [1]

Captured:
Pawn, Pawn, Rook, Bishop, Queen

Figure 22: Status - Scoring (Show Adversary Scores)

Game Over

How the game ends is another key difference between traditional chess and Clausewitzian chess. In traditional chess, the game ends upon checkmate (King Capture is actually prohibited). In addition,

traditional chess has notions of stalemate and draw scenarios (insufficient mating material, threefold repetition, the 50-move rule, etc.). In addition to these standard gameplay ends, the player may also offer a draw or resign (conceding victory).

In Clausewitzian chess, the game over logic is simplified. During gameplay there are only two possible ways for the game to terminate. The first is the standard rule, which is when the game reaches maximum number of turns. For more details, refer to the Number of Turns section. The second is a rare special case in which the game randomly assigns both players the "King Capture" victory conditions. In this case, the maximum number of turns is set to infinity and the game ends as soon as either player captures a King. These are the only ways the game can end from gameplay. Except for in the second rare case, play continues past King Capture and the game does not allow stalemates, draws, etc.

In addition to the two possible ways for a game to terminate through gameplay, there is only one way to end the game outside of gameplay. Players can resign. Offering a draw is not possible. In this game, resigning ends the game without a clear determination of victory. As the game is not zero-sum, there is no inference that if player A resigns, player B automatically wins. Instead, the game assumes that the player that resigned lost. For the player who did not resign, the current convention is to check the status of their victory conditions at the time of resign. We declare the player not resigning as a "possible winner" or "possible loser" as supported by their score status and victory condition since the scores would have likely changed had play continued to the maximum number of turns. Alternatively, if the game is part of an experiment, we disregard the entire game and do not include the data in our findings. Clicking the Logout button does the same action in game as resigning except it brings the player back to the login screen instead of the game lobby. The figure below shows the "Logout" and "Resign" buttons.

Figure 23: Logout, Resign Buttons

The figures below show the resign messages a player would experience if they initiate the resign or receive it, respectively.

Figure 24: Game Over - Initiate Resign

Figure 25: Game Over - Receive Resign

Finally, when the game concludes during normal play, the Score Status section shows the outcome of the game, as shown in the figure below. Both the player and opponent are assigned an outcome (no victory, victory, total victory) based on whether they achieved their victory condition. The status shows the opponent's outcome and scores in brackets.

Figure 26: Game Over - Full Game

Special Rules / Considerations

The inclusion of fog, friction, and piece composition game elements forces some special rules for chess pieces. This section will cover special pawn moves and castling and how they are adapted for Clausewitzian chess.

The first special move that is different from traditional chess is castling. In short, Clausewitzian chess does not support castling. The first reason is that the rules of castling become burdensome and complex when we allow full customization of piece setup. The standard chess layout allows for fixed and straightforward rules for castling. The same rules only apply to some limited cases when setup is customized (notably, King must be between two Rooks). The second reason castling is not supported is that special exceptions would be needed to handle check enforcements and play continuation post King Capture. In traditional chess, players cannot castle if they have moved their King or Rook, if they would be moving their King through at attack, or if their King is in check. Clausewitzian chess does not enforce these rules and the ideal rules and resolution of castling is debatable. Finally, the inclusion of custom setups allows players to generally "castle for free" anyway as they can pre-arrange their pieces as if castled.

The remaining special moves all concern pawns. In traditional chess, pawns have a number of special moves mainly included to speed up the game. First is the initial double pawn move when moving the pawn from its original position. The second is that pawns can attack diagonally only. The third is en passant capturing. Finally, pawns can promote to any major piece (non-pawn).

Piece composition allows players to place pawns in either of the back two ranks. This forces some changes to the special pawn moves. First is the pawn's double pawn move. We still allow this for pawns on the second rank (where pawns traditionally start). The problem is handling the case where a pawn is on the back rank. Some variants allow a double (or triple) pawn move from this location. In the current implementation of Clausewitzian chess, we only allow a single pawn move from the back rank. Subsequently, since the player has moved the pawn, they are unable to perform a double pawn move once they reside on the second rank. The primary reason we include this rule is that is makes it less appealing to "fill out" the rest of the budget by placing pawns in the back row.

The remaining pawn moves remain intact. All pawns still only attack diagonally as in traditional chess. En passant capturing is the same as in traditional chess except that it is only available for pawns that start in the second rank (traditional starting place pawns). Finally, to simplify implementation, we only allow promotion to Queen.

Findings

Our findings are limited to observations made by the game designers and play testers. We did not conduct any experimentation on different parameters or scenarios. As such, the findings discussed here are anecdotal and subject to change. Future efforts should include a more rigorous exploration of how different levels of fog, friction, chance, and asymmetric ends impact player behavior and cognition. In addition, it is possible to run experiments to test specific hypotheses. For example, we could test player performance over different levels of fog to determine if there is an inflection point at which player performance fully degrades regardless of player skill.

During our play testing, we noted three primary findings. First, the fog of war component was reasonable and significantly changed how the players played chess. Second, the friction component was "frustrating" and may even need to be relaxed in future versions. Finally, the asymmetric victory conditions and non-zero-sum outcomes successfully captured the dynamics of competing under conflicting ends, leading to different play styles and strategies for different combinations of victory conditions. In total our play testing incorporated more than 12 unique individuals who played collectively more than 25 games.

The first finding regarding modeling fog of war was important in two ways. First, we should expect players to behave differently in the game under uncertainty. Secondly, their behavior was consistent with real world observations of operating under limited and uncertain information. For example, with full knowledge of the game board, players are more willing to advance their more capable pieces since they fully understand the risks of that action. However, when unsure of those risks, players were likely to hold their more capable pieces in reserve unless they were willing to sacrifice that piece outright. We see this in the real world as well; military operations typically hold back critical assets until the area is secure via maritime superiority and/or air supremacy. We were very encouraged to see player behavior change under our fog of war model and even more so that the new behavior was consistent with real world behavior in some cases.

The second finding noted that while frustrating, friction forced players to change their behavior and gameplay. In this case, the most interesting discussion surrounding this finding is how the friction pieces interrupted the player's cognitive process of selecting appropriate moves. Since this implementation minimized the player's awareness of which pieces the game is holding in friction, players often found themselves considering moves, and eventually attempting to play moves, that involved friction pieces. When the move failed, the players had to adapt their reasoning about selecting their next move. Other approaches to modeling friction may make the friction pieces too obvious, removing the cognitive disruption our approach provides. It may not be an easy balance to strike, but having friction induce some level of frustration on the player resulted in interesting player behavior. In addition, we recommend readers attempt to play a game in which they cannot see either the current and next friction piece. This case is particularly interesting as it is probably the most realistic model of friction available in the game.

The third finding regarding asymmetric ends was important in two ways. First, changing the structure of the game of chess so that it is not zero-sum led to interesting in-game phenomena such as players attempting to deduce their opponent's victory condition and attempting deceitful maneuvers to obfuscate their own. While this kind of behavior is common in the real world, it is certainly not as common in games like chess. This finding is also interesting as it models victory conditions of various difficulty resulting in a game that is not always fair. For example, the "Three-Check" victory condition is often the easiest to succeed at and the most difficult to defend against. The "Most Pieces" and "Most Material" conditions are close in terms of difficulty but are usually slightly more difficult. In addition, they are both much easier to defend against than "Three-Check." This means that if a player can deduce their opponent's victory condition, they can actively frustrate their opponent's efforts. The opposite is also true. Finally, the "King Capture" victory condition models a total war scenario that is significantly more challenging than the other victory conditions. Players often cited a lack of fairness when they had "King Capture" and their opponent had one of the others. This is true in reality; nations have differing

levels of interest at stake in conflicts. Having this emerge from the models was encouraging. We leave teasing out these concepts to future work.

Our conclusion is that we have successfully modeled fog of war, friction, and asymmetric ends within the game of chess. Additionally, we hypothesize that incorporating concepts into known games reduces some barriers for players to learn, exercise, or experience these concepts. This hypothesis, while unsupported in this document, is an interesting one we believe our other efforts support and we leave exploring this hypothesis to future work.

Known Issues

In this section we will attempt to list some of the known issues present in the Clausewitzian chess tool. This list will range from bugs to game elements that could use improvement.

Table 2: List of Known Issues by Category

Category	Name	Short Description
User Interface	Polish	The user interface is rough; utilize human factors and front-end developer expertise.
User Interface	Vague Status	Some of the status messages in the UI are vague and do not provide enough detail. In addition, some status messages overwrite helpful status messages.
User Interface	Improved Dashboard	The current dashboard is limited. Expanding the functionality would add value to those tracking multiple games; for example, clicking on a game and having a person join as an observer would be helpful. Also being able to hide and expand certain games would be helpful.
User Interface	Refactor / Structure	Certain UI elements (forms, page layout) could use refactoring into more flexible and reactive structures so that the game can run on a number of platforms, not just a standard desktop web browser.
Game	Play Continuation	There are game states in which it is a player's turn and they are unable to move. The current resolution is they can either pass their turn or resign. This requires further design.
Game	Pawn Moves	With the custom layout (piece composition), pawns can start in both the back rank and their normal second rank. Pawns have their standard single or double pawn move and en passant only from their normal second rank. How to expand these rules to pawn on the back rank needs design and implementation.
Game	Castling	With the custom layout (piece composition), the game does not support castling. It is unclear what castling rules we should allow based on the minimalist requirements for a valid starting position. This requires design and implementation.

Category	Name	Short Description
Game	Chess Clock / Timed Turns	Add in a traditional chess clock so that players know how much time each has taken in the game. This may allow extensions to where victory conditions can incorporate move time.
Game	Expanding / Generalizing Victory Conditions	Generalize victory conditions such that there are two tiers, easy and difficult. Right now, all victory conditions are easy except King Capture, which is difficult. Generalizing in this way will allow for expanding the available victory conditions in a meaningful way.
Game	Expanding Vision Modifiers	Currently vision is distance-based only with a limited notion of jamming. It would be interesting to expand this beyond distance-based approaches to be able to model different spectrum/domains.
Game	Expand Friction	Currently friction only prohibits a move, simulating a missed or dropped order. Instead, there should be a chance for move perturbation, resulting in a move 1-2 squares off from desired. This requires design, balance, and implementation.
Software	Documentation	The current version is an engineering prototype; we may not have documented all of the code as recommended by traditional open software standards and best practices.
Software	Refactoring / Structure	The current version is an engineering prototype; the code's structure may not be as recommended by traditional open software standards and best practices.
Software	Accessible Chat / Status	Low priority bug. Some users wanted the ability to load chat and status directly to the clipboard for pasting into other applications.
Software	Information History	Some users wanted to be able to view board history. While we do not intend memory to be a core mechanic, there is some dependence on information recall in the current version. Being able to cycle back and see past information would alleviate the memory pressure.
Software	Game Playback Feature	Some users wanted the ability to load a finished game and cycle through moves and perform standard chess analysis on different potential lines of play. This is a standard feature in chess applications. We do currently record the full FEN[3] for each move so that reconstructing any game is already feasible.

[3] Forsyth–Edwards Notation https://en.wikipedia.org/wiki/Forsyth%E2%80%93Edwards_Notation

Category	Name	Short Description
Software	Hooks to AI / Chess Engine	Provide easy accessibility to different chess engines and potential AI engines. Currently the game only supports 2-player human vs human play. Enabling chess engine and AI play would allow single player mode. In addition, providing an easy interface would encourage the development of general chess engines and decision support tools for environments that incorporate fog and friction.
Software	Dashboard Analytics	The dashboard is limited in showing only the current status of ongoing games. Having some functionality in the dashboard to track user progress and performance would be interesting.
Software	Game Spectating	This is a basic functionality in many chess applications that allows people to join the game but not as a player on either side. The software does not currently support spectating in this version.
Software	Database Organization	The current database system (MongoDB) stores records in different data collections. These are currently not very well organized which can result in awkward and inefficient data queries.

On a general note, the tool would benefit significantly from polishing and user interface improvements; unfortunately, this was out of scope for the current effort.

Next Steps

There are some obvious next steps associated with any tool first released in prototype form. This includes continued iteration and improvement of game systems and software implementations as well as tackling the list of known issues. Generally speaking: we divide our next steps into three categories.

The first is bug fixes that exist within the current game. The Known Issues section of this report details most of the major bugs that remain. As with all software, there are likely addition bugs currently unknown. Further testing and bug fixing is a necessary component of future work.

The second category is expanding and improving upon existing game elements. The Known Issues section of this report details most of our desired extensions. These include improving, adjusting, or balancing game systems. In addition, we include in this category engaging with stakeholders in order to 1) understand their feedback of the tool, and 2) prioritize, and define the list of development items needed to improve the existing elements of the game. Finally, as mentioned in the Known Issues section, future work should integrate AI components as both a player assistant (decision support, Course of Action advisor, etc.) and an adversary (super human play, tunable difficulties, etc.).

The third category covers all new items and future directions, including programmatic functions. The highest priority item in this category is the inclusion of human factors experts to help adjust the user interface and cognitive experience to improve our models. The first version of this tools shows that

there are interesting implications on player cognition; future work should inspect this aspect in a more rigorous fashion. Other items left for future work include front-end development (UI polish), running targeted experiments, and testing our hypothesis that building upon known games helps reduce the cognitive burden of players.

MIT Lincoln Laboratory Clausewitzian Chess Team

Joel Kurucar, Game Designer, Game Developer
joel.kurucar@ll.mit.edu

Andrew Uhmeyer, Game Designer, Game Developer
andrew.uhmeyer@ll.mit.edu

Jared Pullen, Game Developer
jared.pullen@ll.mit.edu

Robert Seater, Game Designer
robert.seater@ll.mit.edu

Sean Winkler, MIT LL Team Lead
sean@ll.mit.edu