

Design – Philip Mortimer

Introduction

My system will be used by West Cross Chess Club members to play chess against online players and against a computer opponent. Furthermore, the system will allow approved club members to access the club tab and play their scheduled club fixtures. The system will generate the fixtures and the league table. This problem can be broken down into individual sub-problems which help meet all the objectives and requirements of the system.

To demonstrate this, I have created a hierarchal diagram to represent the decomposition of the task into many sub-problems.

West Cross Chess Club																			
Chess Game			Club Section					Offline Play			Online Play								
2D board with all pieces		Implementation of standard chess rules	System stores leader ID	System displays club contact details	Members can play club fixtures		League Table	Members		Ability to play multiple offline games concurrently	Ability to play against friend offline	Ability to play against computer offline	User log-in		Play against friends		Play against random people	Store users ELO score depending on online performance	Have a continuously running server application that stores user and club details, as well as ongoing game details
	Option for time control customisation	Store previous board states to account for 50 move rule		Club contact details can be changed by leader	Generation of fixtures given list of club members	Each member has a button that allows them to play their fixture	Results from fixtures are collected and league table is updated and displayed	Members can be added to club by the leader	Users can request to join the club	Up to three ongoing offline games are stored locally		Computer uses depth-limited alpha-beta pruned minimax algorithm in conjunction with various heuristics to play chess	User has ability to create account using details such as username and email address	Ability to log-in by providing username or email address and password	Users can play against friends by sharing the six-digit code, which is the game ID, to a	Users can play against friends by typing in their username	Users will be able to play against other players who are looking to find a random online opponent		
									Club leader can approve or deny requests to join the club										

The overall problem needs to be broken down into individual, achievable tasks which are as independent as possible from the rest of the system. This allows for both an effective system design and also a system which fulfils all requirements of each of the sub-elements. I have broken my project down into its main components: the chess game, the club aspect, online play and offline play. For each of these areas, I have decomposed the problem into achievable units which address my objectives.

For ease of reference here are my objectives, specification items and success criteria. Each specification item is denoted by a number. Each objective is denoted by a letter and each success criterium denoted by a roman numeral. A copy of this can also be found in the investigation. When an objective is referenced it may be reference in a form such as 3a.

1. The system will allow users to log in by providing an email address, username, and password. Both the email address and username must be unique.
 - a. To create a database that stores all relevant details, including user details, in the third normal form.
 - i. All data stored must have no repeating groups, partial dependencies or transitive dependencies.
 - ii. Users will be able to log in to the system by providing their email address / username and password. Once logged in, they will be able to view their details such as their ranking or fixtures (if they are a member of the club).
 - b. To check that both the email address and username provided are not already the username or email address of another user. This will be done by searching the file storing all user details.

- i. When attempting to create an account, users who input the email address or username of another user will be unable to use that name and will be asked to choose a different one.
 - ii. When creating an account, a username / email address that has not already been used by another user should be accepted.
 - c. To search both user email address and username to find a match when logging in.
 - i. Users should not be able to create a username that could be a valid email address. This will be achieved by not allowing @ character in usernames and requiring the @ character in email addresses.
 - ii. Users will be able to log-in by providing either their email address or username.
2. Players may choose to play as guests who require no passwords but will have more limited features available to them. Guests will be unable to access features such as the club section but will be able to access offline play.
 - a. When a guest uses the system, they will be given one id that all guests have. This id will be impossible for a normal user to have. With this guest id, the guest will have the ability to play offline games but will not have a ranking or the ability to continue offline games over multiple sittings. Hence, the system will have four access levels (for guests, users, club members and the club leader).
 - i. A guest will be able to play offline but will not be able to access the club section. They will be unable to play games over multiple sessions.
3. There will be an option for users to play online against random people or against friends. People can play against friends by typing in their friend's username or by sharing a random 6-digit code generated by the system. Users will be able to customise the ruleset for online play to allow for correspondence play and playing on a timer.
 - a. The server will have a text file storing all currently ongoing online games. Each time a new game is created, a game id will be generated for it. This will be the six-digit code that users can share. Furthermore, it will store details such as the board state and the user id of players, as well as the ruleset.
 - i. Users will be able to play against friends either by sharing a six-digit code or by typing their username. The ruleset for these games will be customisable.
 - b. Players will be able to play against random people by pressing a button. They will be matched against another user who also wishes to play online. If nobody wishes to play online, the system will tell the user that no match could be found. This will be achieved by having the server store the user id of all players who wish to play against a random person. When two ids are found, they will be matched and have a game started.
 - i. Users will be able to play against random opponents when they can be found. If no opponents can be found, an appropriate message will be displayed.
4. Users will be able to play chess locally against each other with users using the same computer to play against each other.
 - a. A file containing all ongoing offline games will be stored. This will store up to three current games. When a file is clicked on, the game is resumed.
 - i. Users will be able to play up to three offline games at a time over multiple session.
 - b. Users will be able to play offline and customise the time controls. Each offline game will be given a game id.

- i. When creating a game, users can choose which time control they would like to use. This time control will then be used for the game. This includes the option for no time restrictions.
5. Users will be able to play against a computer of 5 levels of varying difficulty, which should range from easy to challenging.
 - a. I will use a depth limited alpha-beta pruned minimax algorithm to deduce which moves should be made. The varying difficulty will come from use of heuristics of different complexities in combination with different search depths.
 - i. I will play each of the computer algorithms and rank them in order from easiest to most difficult. Then I will have the computers play each other. If my ranking is the same as the ranking obtained from playing the computers against each other, I will have clearly succeeded in creating five distinct difficulty settings.
6. When playing against the computer or locally, the user should be able to customise timer settings and decide who is what colour.
 - a. Create a visual menu that allows the user to select from a range of a time controls, including unlimited and the FIDE official timer control settings. Users will also be able to decide what level of computer to play against and whether the computer is white, black or random. These settings will be stored, and a new game will be created
 - i. A user should be able to play local games using various different time controls and against a computer of varying difficulty. This can be verified by testing how long it takes for a user to lose due to running out of time.
7. The system will have a West Cross Chess Club section where Lewis James will have the ability to add, edit and delete member details.
 - a. The server will store details of all club members in a table that is in the third normal form. Lewis James will be able to retrieve and view this information. He can then edit user details, which will be updated on the server. He can also remove users from the club or add users to the club. This will be achieved by updating the file on the central server.
 - i. The club leader will be able to view all member details and edit these. The leader will also be able to remove or add members to the club. This can be shown to have been successful by viewing club details after changes have been processed.
8. Users will be able to request to join the club, a request which the leader can either approve or reject.
 - a. Users who are not a member of the club will click on the club section and have the option to send a request to join the server. If they choose to send the request, the request will be sent to the server. The server will then store this request. The leader, currently Lewis James, will then have the option to allow these members to join the club or to reject the request.
 - i. Users will be able to request to join the club and Lewis James will be able to see these requests. Lewis James will then either approve or deny these requests. This can be verified through testing by sending a request and seeing if it shows up for the leader. Then, the leader will accept the request. If the user who requested to be a member is now part of the club, the objective will have been fulfilled.

9. The West Cross Chess Club section will generate the fixtures and users use this section to play their allocated games against each other. The computer automatically gathers the results from the scheduled games and updates the league table accordingly.
 - a. The computer will use the list of current members to generate a fixtures list for the season. If new members join the club halfway through the season, the remaining fixtures will have to be updated to allow the new member to play.
 - i. If the system generates fixtures in a way that each user plays each other exactly twice and so that every game occurs every other Tuesday, the algorithm will have succeeded.
 - ii. If new members join the club when there are 3 games (for example) left of the season, the new member should still be able to play those remaining three games.
 - b. The user will be able to view their fixtures in the club section. They will then start a new game with the member they are scheduled to play against for that week.
 - i. The user will view their fixture in the club section and be able to play the game there. The result of the game will be collected and lead to the league table being updated.
 - c. The system will collect all results and use them to update the league table standings. The league table, which is stored by the server, can be viewed by players.
 - i. Once fixtures have been played the league table should be updated to display the new scores and standing.
10. When all the club league games have been played, the system displays the winner and resets the table and fixture list.
 - a. When the system reaches the end of the current fixture cycle, the league table is displayed along with a message that shows who won. The table is reset, and the fixture generation algorithm is run again to start a new season.
 - i. When the end of the season is reached, a message should be displayed to indicate that the season is over.
 - ii. When the end of the season is reached, the table should be set so that everyone has zero points and the fixtures should be generated once more.
11. Lewis James will have the option to make another user the club leader if he no longer wishes to be in charge.
 - a. The system will store the user id of the club leader. If Lewis James wishes to make somebody else club leader, he will need to know their username. The system will then set this user's id as that of the club leader. This will allow the Lewis James to make somebody else the club leader.
 - i. The current club leader will be able to transfer the role of club leader to another valid user. If they do choose to transfer the role, the user who they have transferred the role to will be able to perform all the tasks that only the club leader can perform (such as updating the club's contact details).
12. The chess game will feature a two-dimensional chess board which allows users to move by clicking on a piece and then choosing from all available moves. The interface should be visually pleasing and clear and should highlight available moves and which piece has currently been selected.
 - a. To create a GUI using built-in features such as JFrame to display a visually pleasing chess board. The view will be a top-down view of a two-dimensional board and will be similar to chess.com's board. The control scheme must be very usable and intuitive.

- i. The board must feature a two-dimensional top down view and be visually pleasing. To test this, I will ask five members whether the board is visually pleasing and looks somewhat similar to typical two-dimensional top down boards (such as Lichess' board). If all five members agree, I will have succeeded. I will also ask the members whether the control scheme is usable and intuitive also.
 - b. When a piece is selected, the piece is highlighted, and the system calculates all possible moves for the piece to make. These possible moves will be highlighted.
 - i. To test this, I will select each piece across multiple games and use my knowledge of chess to determine whether all legal moves have been identified.
- 13. The menu system should be clean and visually clear, containing a small range of options which allow the user to perform the tasks they wish to.
 - a. I will use the built in NetBeans drag and drop GUI builder to create a clear and visually pleasing menu. The menu will be simple and contain a small range of buttons that allow the user to navigate to the desired section of the system.
 - i. Navigation should be intuitive and easy, even for novice users. Text and buttons should be readable and large. To measure this, I will ask five members whether they find navigation easy and whether they find the menu clear. If all five agree, I will have succeeded.
- 14. The system will have a constantly running server application which handles online play and user log in. The server will store all member details, as well as West Cross Chess Club details such as the table as well as a text file containing all ongoing games. The system will communicate with the server to facilitate online play.
 - a. The server application will be constantly running and will handle all elements of the club that are online. This includes the creation of accounts, user log-in, online play, storing chess club details, collecting fixtures results, updating the league table, storing the chess club leader and scheduling fixtures for the chess club.
 - i. If players are able to log-in and perform online tasks such as creating an account or playing against a friend online, the objective will have been met.
- 15. The system will feature contact information for the club and the leader (currently Lewis James) will have the ability to update this information.
 - a. There will be a section in the club tab, which will be accessible to all users connected to the internet (including non-club members), which displays the email address and phone number of the club. These details will be stored on the server and the club leader will have the opportunity to update these details.
 - i. All users should be able to view the club's contact details and Lewis James should be able to change these contact details.
- 16. The system will rank players via their online play results using a ratings system known as ELO.
 - a. After each online game, the server will receive the result and calculate the new ELO score for each player following a set of formulae defined earlier in this document. The ELO rating of a player will be stored by the system and displayed alongside username when playing an opponent.
 - i. After a game against an opponent, I will calculate what each user's new ELO score should be and compare it to what the system determined they should be. If these scores equate, the system will have succeeded.
- 17. When playing online, the system will display both users' username and ranking.

- a. When matching against an opponent, the server will store both users' name and ELO rating. These will be displayed to the other user when playing online. These (and all) calculations should be correct and accurate.
 - i. When playing online, users should be able to see both their username and ELO rating as well as their opponent's rating and username.

18. GENERAL OBJECTIVES AND CRITERIA

- a. The system should be easy to maintain. I will achieve this through use of extensive documentation (e.g. comments in code to explain the purpose of methods). Bugs should be easy to identify and fix, something which I will achieve by designing the system in a modular fashion.
 - i. A competent programmer should be able to recreate the system given its documentation.
 - ii. My system will be modular, making use of many independent methods. To test whether my system is easy to debug, I aim for users to encounter no more than three minor bugs in a session and no major bugs in an average session.
- b. The system should fully meet the requirements of the specification.
 - i. I will ask Lewis James whether the system performs as he intends it to once I finish it.
- c. Validation should be employed for all appropriate fields of manual data entry.
 - i. I will aim to catch at least 90% of invalid inputs.
- d. All calculations performed should be accurate and correct
 - i. Calculations such as ELO rating should be accurate and all relevant success criteria relating to ELO calculation should be achieved. All other calculations should be correct, such as the updating of the league table.
- e. The system should be visually clear and easy to navigate. It should be easy to use, even to those unfamiliar with technology.
 - i. I will ask chess club members whether they found the system intuitive and easy to use, and if they all agree, I will have succeeded.

Generally, all sub-problems link to an objective and the hierarchal diagram essentially illustrates how I will achieve my objectives.

I have broken down the chess game into the creation of a 2d board with all the pieces, something which links with objective 12a, which discusses the creation of a visually satisfying 2d chess board. It also links to 18e, which discusses the need for general visual clarity. The implementation of standard chess rules including the option for time controls links to objectives 6a and 4b, both of which describe the ability of users to customise timer controls. It also links to a wide range of objectives which discuss the ability of users to play chess off and online, as these will require implementation of all chess rules.

My diagram shows that I will also have a section of the system devoted to the West Cross Chess Club. In this section, the system will store the ID of the leader, who will be able to change and display the club's contact details as well as deciding which users can be members of the club. This addresses objectives 8a and 7a, which state that the leader should be able to delete and add members to the club in addition to approving and rejecting requests from users to join the club. I have also identified that the system needs to generate club fixtures and update a league table after

these fixtures have been played. This directly links to objectives 9a, 9b and 9c, all of which state that the system should handle fixture generation and result collection.

For offline play, the system will store up to three ongoing offline games at a time. This address objective 4a which talks about having a text file that stores three ongoing offline games. The details of each game as well as the current board state will need to be stored to allow users to resume play. The ability to play against a friend or a computer locally addresses objectives 4a and 4b. The creation of a computer of varying difficulty levels to play against offline links with objectives 5a and 14b, which discuss the need for a computer of 5 difficulty levels and also the need for all calculations performed to be accurate. In this case, the minimax algorithm needs to be implemented correctly to ensure sound decision making from the computer.

For my online play segment, I identified numerous sub tasks to perform. For most online features, a user has to log-in. This is achieved by providing their email address and password. To create an account, they also need to provide similar details. These details need to be appropriately validated. One form of valid log-in is logging in as a guest. These problems will help me meet objectives 1a, 1b, 1c and 2a. Online play against friends is supported, which can be done either by sharing a six-digit code or by typing in a friend's username. This links directly to objective 3a which talks about the server storing all ongoing online games including allowing for players to play via the two aforementioned methods. The system will also allow play against random people, something which will be achieved by matching against any player also requesting a random game. This sub-problem will address objective 3b. I have also identified the need to calculate the ranking of players using a rating system known as ELO. This will be achieved by analysing results of online games. This links to objectives 18d, 16a and 17a, which discuss the need for accurate calculations and for these calculations (including the ELO score) to be displayed.

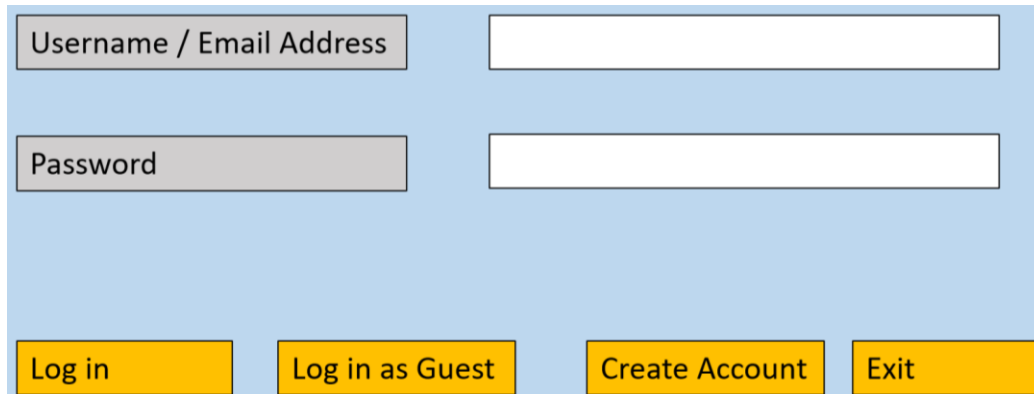
Here is a landscape copy of the diagram (in case the other copy is too small)

West Cross Chess Club													
Chess Game		Club Section				Offline Play		Online Play					
2D board with all pieces	Implementation of standard chess rules	System stores leader ID	System displays club contact details	Members can play club fixtures	League Table	Members	Ability to play multiple offline games against concurrently	Ability to play against friend offline	Play against random people	Play against random people	Store users ELO score depending on performance	Have a continuously running server application that stores user and club details, as well as ongoing game details	
	Option for time control	Store previous board states to account for 50 move rule	Club contact details can be changed by leader	Generation of fixtures given list of club members	Each member has a button that allows them to play their fixture	Results from fixtures are collected and league table is updated and club by the leader	Members can request to join the club	Up to three ongoing offline games are stored locally	Computer uses depth-limited alpha-beta pruned minimax algorithm in conjunction with various heuristics to play chess	User has ability to create account using details such as username and email address	Ability to log-in by providing username or email address and password	Users can play against friends by sharing the six-digit code, which is ID, to a	Users can play against other players who are looking to find a random online opponent
Club leader can approve or deny requests to join the club													

Input and Output

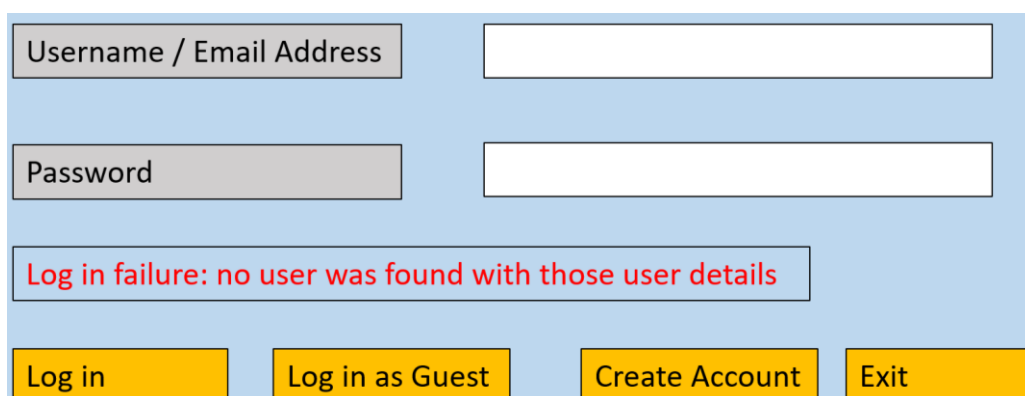
In this section, I will describe all inputs and outputs in my system, as well as designing these inputs and outputs. I will consider and justify why data is included in the system's outputs.

Log-in form:



The log-in form consists of a light blue rectangular area. At the top, there is a label 'Username / Email Address' in a grey box next to a white input field. Below this, there is a label 'Password' in a grey box next to another white input field. At the bottom of the form, there are four yellow buttons arranged horizontally: 'Log in', 'Log in as Guest', 'Create Account', and 'Exit'.

This form shows what I plan for my log-in system to look like. The system requires a password and either a username or an email address to log-in. The option for either an email address or a username is included as many find it easier to remember their email address than a username. There are also three buttons for the user to perform certain actions. The first button ("Log in") allows the user to log in using the inputted details. This sends a request to the server to see if any users have the same details as inputted. If a user does have matching details, the log in will be successful. If no matching details are found, an error message will be displayed. Users will also be able to press the enter button to log in, however less technologically confident users may find a button more intuitive. The log in as guest button will allow the user to access features available to a guest, such as offline play, and requires no username or password to use. The guest log-in is important as it will allow users to access some of the system offline, a major advantage over websites that offer chess systems. The create account button will take the user to a form which allows them to create their own account. If the username and password do not match or are invalid, an appropriate error message will be displayed. An error message may look something like this:



This is the same log-in form as above, but it now includes an error message. The error message is 'Log in failure: no user was found with those user details' in red text, enclosed in a white box with a grey border. The buttons remain the same.

This error message is important as it shows the user that their log-in details are invalid. This form helps fulfil objectives 1c and 2a (as it allows users to log-in using their details or as a guest) and it will take users to a form to create an account which helps address objectives 1a and 1b.

Account Creation form:

Create Account - * indicates required field

Username *	<input type="text"/>	
Email Address *	<input type="text"/>	
Password *	<input type="password"/>	
Confirm Password *	<input type="password"/>	
First Name	<input type="text"/>	
Surname	<input type="text"/>	
Post Code	<input type="text"/>	
Address Line 1	<input type="text"/>	
Address Line 2	<input type="text"/>	
Address Line 3	<input type="text"/>	
Address Line 4	<input type="text"/>	
Discord Username	<input type="text"/>	
Discord Number	<input type="text"/>	
Date of Birth	<input type="text"/>	
Phone Number	<input type="text"/>	
Create Account	Clear Form	Back to Log-in

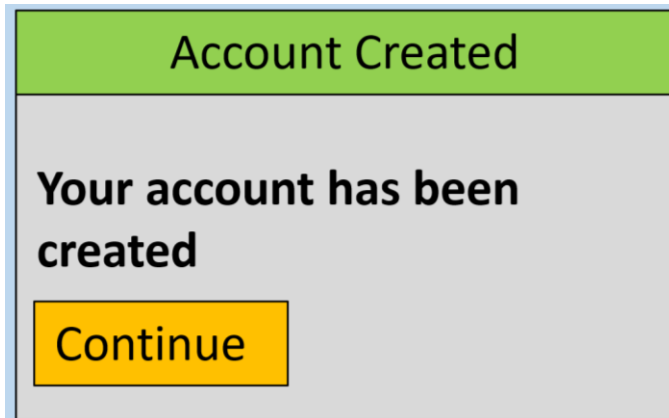
This form is used by users who wish to create an account. Once all fields have been filled in correctly (and validated), if the create account button is pressed, the server will process the request to create a new user account. If both the email address and username are unique to the user, a new account will be created with the inputted details. Thus, this form will help achieve objectives 1a and 1b (by allowing a user to create an account that is unique). If the clear form button is pressed, all fields will be set to empty once again. If the back to log-in button is pressed, the user will be taken back to the log-in form. If the create account button is pressed and an error is detected, the user will be shown an appropriate message:

Account Creation Failed

**Account Creation Failed:
Username inputted already exists**

Continue

Equally so, if the account is successfully created, an appropriate message will also be displayed.



It is vital that the system communicates clearly to the user to avoid ambiguity and uncertainty. If an account is successfully created, the user will be taken back to the log-in form. The first three fields are required as a password is necessary to prove identity and both an email address and username are asked for to reduce the chance of a user being locked out of their account due to forgetting their log-in details. Whilst there are only three required fields, there are many more voluntary fields which ask for rather sensitive data. These fields were included because the club leader, our client, requires these details in order to run the club. Address and contact details are used to coordinate meetings and in case of emergencies (such as a medical emergency), Discord details are used to coordinate voice chat when using online play and date of birth is used for safeguarding purposes. All of these details are stored by the current system and hence Lewis James wishes to carry them across. Failure to input these details will not prevent members from joining the club, however all members should have the option to put these details in to help the club function as effectively as with the old system. And users who are not club members will have no need to input these details. As these details are sensitive, these details will not be accessible to other users. However, the club leader (currently Lewis James) will be able to view and even alter these details. Therefore, users will be warned that all details (other than password) will be visible to the club member if they join or added to the West Cross Chess Club. The warning will also make clear that these details are not required, even to join the club, although they are desirable. This warning will also help users understand what their sensitive and non-sensitive data may be used for.



A warning dialog box with a yellow header bar containing the word "Warning". The main text area is grey and contains a detailed warning about account details being viewable by the West Cross Chess Club leader and other members. At the bottom, there are two buttons: a red "Cancel" button and a green "Continue" button.

Warning

All account details other than password will be viewable by the West Cross Chess Club leader if you join the club or are added to the club. If you are comfortable with this, please fill out all fields. However, you are not required to fill any fields out other than username, email address and password, even if you wish to join the West Cross Chess Club. If you press "Continue" you agree to all inputted details other than your password potentially being viewable to the West Cross Chess Club leader (who can change some of this information) and your username and ELO rating being visible to online opponents. You also agree to your first name, surname, results and username being viewable to all West Cross Chess Club members if you join the club. If you wish to go back or change your form entry, please press "Cancel".

Cancel Continue

This warning is lengthy but highly necessary to ensure that the user's data is used fairly, legally and ethically.

Main Menu Form:



A main menu form titled "West Cross Chess Club" on a light blue background. It features three grey buttons stacked vertically: "West Cross Chess Club Section", "Offline Play", and "Online Play". At the bottom, there are two yellow buttons: "Go to Log-in Page" and "Exit".

West Cross Chess Club

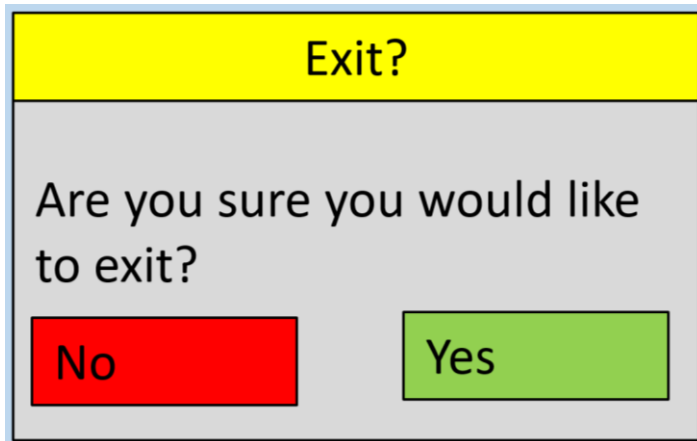
West Cross Chess Club Section

Offline Play

Online Play

Go to Log-in Page Exit

This form is my main menu. The exit button allows the user to exit the system. This only occurs after the user confirms that they wish to exit:



A dialog box with a yellow header bar containing the text "Exit?". Below the header, on a light gray background, is the question "Are you sure you would like to exit?". At the bottom of the dialog are two buttons: a red button labeled "No" and a green button labeled "Yes".

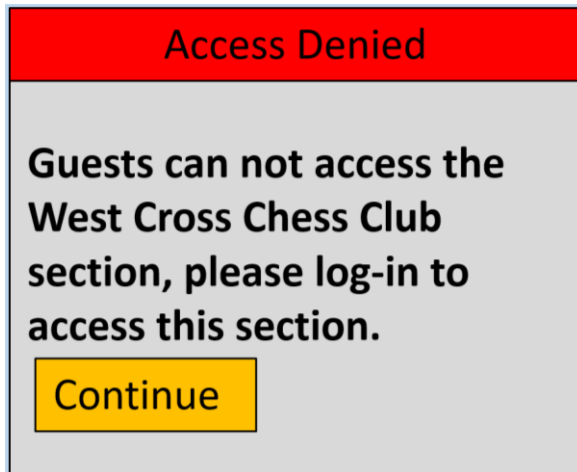
The main menu is very clear and only has three main options. This is so that users find navigation easy and can easily access the elements of the system that they wish to use. This will address objectives 13a and 18e, both of which talk about ease of navigation, particularly on the menu. This menu will allow users to access the core features of the system. In this segment, the data collected is simply the button pressed, which will result in the user being taken to a form of their choosing. Users who press the "Go to Log-in Page" button will be logged out and taken back to the log-in form. Each of the other three buttons will take the user to the relevant area of the system. These buttons roughly correspond with the broader sub-problems I have identified for the system. In this section, the inputs are the given button presses and the outputs are the user being taken to the appropriate form. Pressing the "West Cross Chess Club Section" button takes the users to different forms depending on their access level. There are four access levels: guest, user, member and leader. I have outlined how the access level is calculated in pseudocode later on in the design.

West Cross Chess Club Section Form for users who are not club members:



The form is titled "West Cross Chess Club Section" in a dark blue header. It is divided into two main columns. The left column is titled "Contact the Club" and contains two input fields: "Email Address:" with the value "exampleemail@gmail.com" and "Phone Number:" with the value "01792 401265". The right column is titled "Club Membership Status" in a yellow header and contains the text "You are currently NOT a member of the West Cross Chess Club". Below this text is a yellow button labeled "Request to Join The West Cross Chess Club". At the bottom center of the form is a yellow button labeled "Main Menu".

The West Cross Chess Club section will have multiple access levels due to the different requirements of different users. This form shows what the section looks like for logged-in non-guest users who are not currently members of the club. If a guest attempted to access this section, they would be met with an appropriate error message:



Access Denied

Guests can not access the West Cross Chess Club section, please log-in to access this section.

Continue

A dialog box with a red header bar containing the text 'Access Denied'. The main body is light gray and contains the text 'Guests can not access the West Cross Chess Club section, please log-in to access this section.' in bold. At the bottom is a yellow button labeled 'Continue'.

They have the option to contact the club, a requirement of the system. The members also have the option to go back to the main menu, something which is needed to enable navigation across the system. When requesting to join the club, members are met with a message asking them to confirm that they would like to request to join the club:



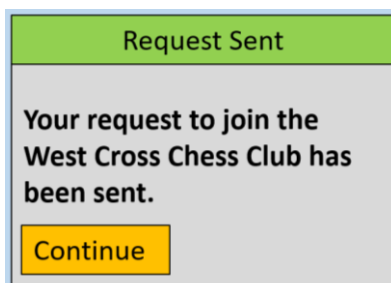
Request to Join?

Are you sure you would like to request to join the West Cross Chess Club?

No **Yes**

A dialog box with a yellow header bar containing the text 'Request to Join?'. The main body is light gray and contains the text 'Are you sure you would like to request to join the West Cross Chess Club?' in bold. At the bottom are two buttons: a red one labeled 'No' and a green one labeled 'Yes'.

Requesting to join causes the user to send their user id to the server, where the fact that the user wishes to join is stored. The club leader can approve or deny this request. When the request is processed, a confirmation message is displayed.



Request Sent

Your request to join the West Cross Chess Club has been sent.

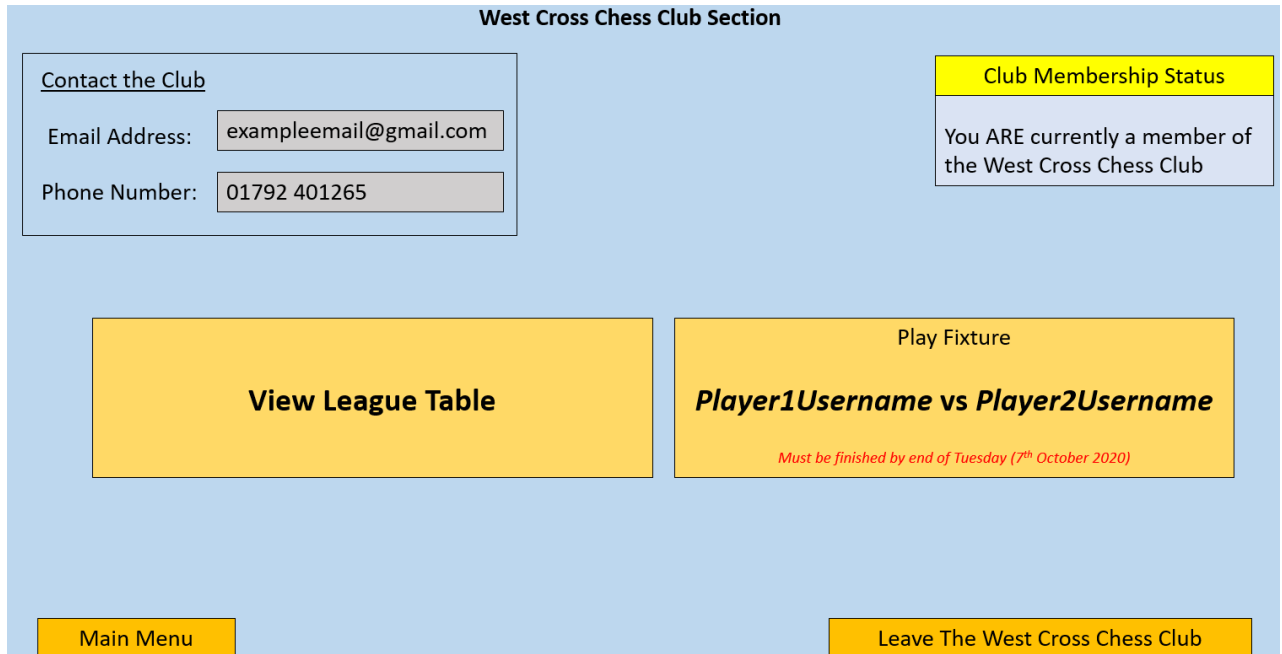
Continue

A dialog box with a green header bar containing the text 'Request Sent'. The main body is light gray and contains the text 'Your request to join the West Cross Chess Club has been sent.' in bold. At the bottom is a yellow button labeled 'Continue'.

These forms satisfy objectives 8a and 15a, which talk about the ability of all non-guests to view club contact details and the ability of non-guest, non-member users to request to join the club. Both email address and phone number are displayed as it is common for a person to not have an email address or a phone number, but it is very uncommon for a user to have neither. It also allows users

to use their preferred contact method, with email being useful for textual queries and phone being better for more complex queries.

West Cross Chess Club section form for club members that are not the club leader:



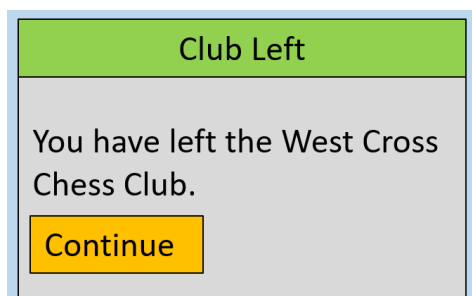
The form is titled "West Cross Chess Club Section" and is set against a light blue background. It contains several interactive elements:

- Contact the Club:** A section on the left with two input fields. The "Email Address:" field contains "exampleemail@gmail.com" and the "Phone Number:" field contains "01792 401265".
- Club Membership Status:** A yellow header box on the right containing the text "You ARE currently a member of the West Cross Chess Club".
- View League Table:** A large yellow button in the center-left.
- Play Fixture:** A yellow button in the center-right with the text "Player1Username vs Player2Username" and a red note below it: "Must be finished by end of Tuesday (7th October 2020)".
- Main Menu:** A yellow button at the bottom left.
- Leave The West Cross Chess Club:** A yellow button at the bottom right.

This form is for chess club members. It gives users the option to leave the club or go back to the main menu. Users who wish to leave the club will have a confirmation message asking them to confirm whether they wish to leave. If they choose to leave, the server will update the list of club members to reflect this.



A confirmation dialog box with a yellow header "Request to Join?". The main text asks, "Are you sure you would like to leave the West Cross Chess Club?". At the bottom, there are two buttons: a red "No" button and a green "Yes" button.



A confirmation dialog box with a green header "Club Left". The main text states, "You have left the West Cross Chess Club.". At the bottom, there is a single yellow "Continue" button.

This features similar data to non-members, displaying contact details and current membership status: items useful for contacting the club leader and for clarity. However, the two big buttons in the middle are the primary functions users wish for the system to perform. The two buttons are big

to enable ease of navigation and visual clarity (something required by objective 18e). The view league table button requests the league table from the server and displays it in a new form. Similarly, the play fixture button allows the user to play their fortnightly fixture. The user can use this button to make moves in their game. The date by which this fixture needs to be played by is also displayed, something which is important to ensure that users play all of their fixtures. The usernames of both players involved is displayed to enable users to talk to each other via external software such as Discord if they wish to. Pressing the button to play the fixture results in a form identical to an online chess game form. This form links to objectives 9b and 9c, which require for the user to be able to play their club fixtures in the club section and to be able to view the league table.

League table form:

West Cross Chess Club League Table				
Player	Wins	Draws	Losses	Points
Steve Stewart	8	0	2	24
Kelly Holmes	7	2	1	23
Amy Rees	6	1	3	19
Jen Williams	5	2	3	17
Ryan Allen	5	2	3	17
Lewis James	5	1	4	16
Ali Trippier	4	3	3	15
Hannah Marie	4	2	4	14
Mary Evans	3	2	5	11
Lilly Jones	2	1	7	7
John Lockhart	2	2	6	6
Jack Grove	0	2	8	2

Back

This form will show all club members name (or username if no name has been inputted), along with their points tally and corresponding ranking. I have included the number of wins, draws and losses as the old system also records these statistics. The back button takes the user back to the club section. This table meets objective 9c and is updated and stored by the server, which collects the results from all played league games. The names are included (when they are provided) because this is how the current system operates and because it is easier for club members to know which person is top (as usernames do not always provide a good indication of who a person is). However, the system may be forced to display the username instead of the name as users are not required to enter their name.

Club game form:

8	R	KN	B	Q	K	B	KN	R
7	P	P	P	P	P	P	P	P
6								
5								
4								
3								
2	p	p	p	p	p	p	p	p
1	r	kn	b	q	k	b	kn	r
	a	b	c	d	e	f	g	h

Timer:
26:12

Black
Username: *username1*
ELO Score: *eloscore1*

White
Username: *username2*
ELO Score: *eloscore2*

[Main Menu](#)
[Resign](#)
[View Previous Board State](#)

This form is typical for a standard game of chess in the club section. A user may resign, causing them to forfeit the game. A user may exit to the main menu (with the game considered being paused). The button to view the previous board state is included for players who forget what the most recent move played was. If this option is pressed, the button will be renamed to “View Current Board State”. Both player usernames are displayed, along with their ELO ratings (fulfilling objective 17a). Both of these data items are included to give the experience a personal feel (that can’t be obtained when playing against a robot) and also to gauge the skill level of the opponent. The timer is needed to display the amount of time left. This will be paused if a user leaves the club section. To make an input, the user clicks on a square of the board. The system then highlights all available moves from a given square. From these legal moves, the user selects one to choose as their move. Once a move is made, a message is sent to the server to communicate that a move has been made. The other user will periodically request for updates from the server, to see if a move has been made. If a move has been made, the other user updates the board to reflect this.

Leader club form:

The club leader has similar options to other club members as they can play their fixtures and view the league table from the section. The email address and phone number fields are editable by the leader. If “change contact details” is pressed, these new details are sent to the server which changes these values in the club file. The membership status of the user is also displayed. The leader also has the option to make another user the club leader, something which may be used if the leader quits the club. This form facilitates the completion of multiple objectives including objectives 7a, 8a, 11a and 15a (which talk about all the things the club member can do, such as altering the club’s contact details). This form does contain many buttons; however, this is acceptable as this form will only be visible to the leader, who will have to have a degree of technological competence. The approach of having many buttons on one form is preferable to having a smaller amount hidden in sub forms as this makes navigation less intuitive.

Appoint new leader form:

This form allows the leader to appoint a new leader (in the current leader’s place). If the user is not found, or an error occurs, an appropriate error message will be displayed. If the appoint button is pressed, the server will search the user table to see if the user exists and if they do, they will be set as the new club leader. This allows the system to satisfy objective 11a. The system has the ability to

appoint a new leader as the old leader may wish to no longer be part of the club, necessitating a new leader to continue effective use of the system.

View all member details:

Username	Email Address	Phone Number	First name	Surname	Postcode	Discord username	Discord number	Date of birth	Address Line 1	Address Line 2	Address Line 3	Address Line 4

This form is accessible by the club leader and allows for all club member details to be displayed. This is used to mimic the old system, which also supported this. This information is useful as it allows the leader to see all members, which could be used when organising sessions. This data is only visible to the club leader and all users have consented to sharing this. This data is collected as the leader says that it is needed to ensure that the club can function properly. This form will display the details of all users in the club by searching the user file for club members. This form allows the system to achieve objective 7a.

Edit users form:

Username	<input type="text"/>
Email Address	<input type="text"/>
First Name	<input type="text"/>
Surname	<input type="text"/>
Post Code	<input type="text"/>
Address Line 1	<input type="text"/>
Address Line 2	<input type="text"/>
Address Line 3	<input type="text"/>
Address Line 4	<input type="text"/>
Discord Username	<input type="text"/>
Discord Number	<input type="text"/>
Date of Birth	<input type="text"/>
Phone Number	<input type="text"/>

Find Member to edit details of

Clear Form

Back to Menu

Update user details

This form allows the club leader to retrieve a specified member's details onto the form and edit them. The find member button allows the leader to search for a member using their username. If a user is found, their details are loaded onto the form. The club leader cannot edit either email

address or username. If “update user details” is pressed, the record of the user is updated to reflect the changed details. This allows the system to fulfil objective 7a (editing club member details).

Add and remove members form:

The form is titled 'Add and remove members form:' and is set against a light blue background. It contains a series of input fields for user details: Username, Email Address, First Name, Surname, Post Code, Address Line 1, Address Line 2, Address Line 3, Address Line 4, Discord Username, Discord Number, Date of Birth, Phone Number, and Club Membership status. Below the input fields are three yellow buttons: 'Search for user', 'Clear Form', and 'Back to Menu'. At the bottom left, there is a yellow button labeled 'Add user to club'.

This form allows the leader to search for a specific user and load their details into non-editable text boxes. The leader can then add or remove the user from the club. The add button will not be visible or usable until a record is searched for. The button will display “Add user to club” if the user searched for is not a member and will display “Remove user from club” if the user is a member. This data will be displayed to reduce the chance of the leader adding/removing the wrong person due to mistaken identity. Once more, this helps fulfil the requirements of objective 7a. The user will be searched for by entering their full username.

Approve / deny requests to join club:

The form is titled 'Requests to join The West Cross Chess Club' and is set against a light blue background. It contains a series of input fields for user details: Username, Email Address, First Name, Surname, Post Code, Address Line 1, Address Line 2, Address Line 3, Address Line 4, Discord Username, Discord Number, Date of Birth, Phone Number, and Club Membership status. Below the input fields are three buttons: a red 'Deny request' button, a green 'Approve request' button, and a yellow 'Back to Menu' button.

This form allows the club leader to confirm and reject requests to join the West Cross Chess Club. A user will have their record displayed in the form and the leader can press approve or deny. If approve is pressed, the user becomes a club member. If deny is pressed, the user is not allowed to be part of the club. If there are no requests, an appropriate message will be displayed. This will directly satisfy objective 8a which specifies that the club leader can approve requests to join the club.

Those are all the relevant forms for the West Cross Chess Club section. Next, I shall display all relevant forms for online play.

Online play menu:

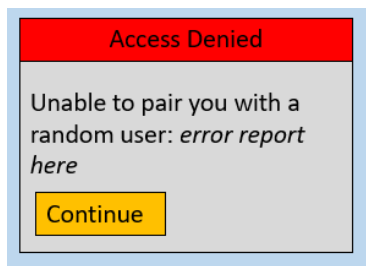
The screenshot shows a light blue rectangular area titled "Online Play" at the top center. Below the title are three buttons stacked vertically: "New Game Against Random Player", "Create New Game Against Friend", and "Play Game Against Friend – Enter Game Code". Below these buttons is a table with four columns: "Slot Number", "Opponent", "Time Control", and "Game Code". The first row of the table contains the values "1", "opponent1", "Unlimited", and "124231". Below the table is a yellow button labeled "Menu".

Slot Number	Opponent	Time Control	Game Code
1	opponent1	Unlimited	124231

This form provides all of the key online options: creating and playing games. It also allows users to go back to the menu. A player may have up to three non-club online games going on at any moment. These three games are displayed in the menu. A user may click on one of these games to resume playing that specific game. Detail such as username and time control are included to allow users to differentiate between their ongoing online games. Slot number is included to aid ease of navigation and visual clarity (something required by objective 18 e). Game code is displayed as this is the code that can be shared with friends to allow them to play with you. Furthermore, there are three buttons that allow the user to execute all required online functionality. Each time a new game is created, it is added to the table displaying all the games. A user can only play three games online at any one time. Users will have the option to resign a game. Once a game is resigned (or a natural checkmate or stalemate is reached), the game will be deleted from the server. This form helps address objectives 3a and 3b, which talk about online play features such as playing against random opponents.

The screenshot shows a message box with a green header bar containing the text "Game Found". Below the header bar is a grey rectangular area containing the text "You have successfully been paired against a random opponent." At the bottom of the grey area is a yellow button labeled "Continue".

If a random game is found, an appropriate message will be displayed for visual clarity.



Access Denied

Unable to pair you with a random user: *error report here*

Continue

Similarly, if no match is found, an error message will be displayed. This could be due to an error such as a faulty internet connection, however it may also be because no other users requested to play against a random opponent (something probable given the small system audience).



Create Game Against Friend

Time Control	15 -10 (Fast)	40-15 (Normal)	Unlimited
--------------	---------------	----------------	-----------

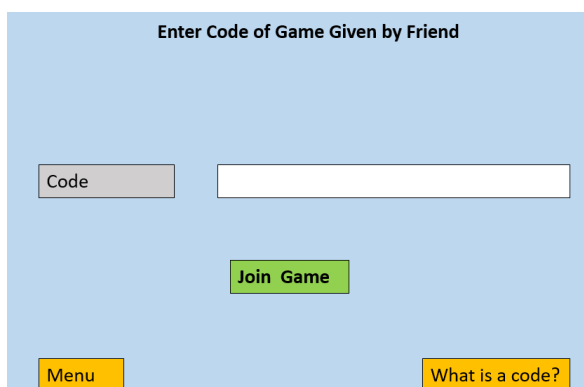
You play as	White	Random	Black
-------------	-------	--------	-------

Username of Friend

Share Game Code Instead? Yes No

Menu Create Game

This form will be used to create the game. This will be validated so the user cannot select yes for “share game code instead?” and have a non-empty username field. This is to minimise the chance of confusion from the user. The user has the option to alter time controls as well as which colour, they play as, something which meets objectives 3a. Furthermore, the general ability to set up an online game using either a code or a username also is required by objective 3a. I feel that the form is very clean and shows a minimal amount of data, something which should make navigation easier.



Enter Code of Game Given by Friend

Code

Join Game

Menu What is a code?

This form allows users to enter the six-digit code a friend may have shared with them to allow them to play against each other. This can be used as an alternative to knowing a friend’s username. I have also included an option for the user to find out what a code is as this is the part of the old system that I personally found most confusing when investigating. This section fulfils the requirements of objective 3a, requiring the ability to play a game by sharing the game code. Both the code field and

username field will be validated to ensure that only valid inputs can be made. They will be validated by first checking to see that they are of the appropriate format and then by searching for the appropriate user / code.

What is a code?

Whenever a new game is created, a unique 6 digit will also be created. If a friend has started a game, they can invite you be entering your username. However, they can also choose to share a six digit code with you. If you have that code, please enter it in the text field to join the game.

Where can my friend find the code?

If your friend logs in, they should press the "Online Play" button. From here they will see a table that shows all the games they are currently playing. The game code of each of these games is displayed in the furthest column. If there is no username in the second column, this indicates that this is a game that can still be joined by typing in the code.

Ok

This form appears when the "What is a code?" button is pressed. I don't think that this form is necessary, however this part of the system may be confusing to users and hence I have provided some information in case users struggle to understand the system. This meets my objectives regarding usability (18e).

The form for playing an online game is identical to the form for users playing an online club game:

Timer:
26:12

Black

Username: *username1*

ELO Score: *eloscore1*

White

Username: *username2*

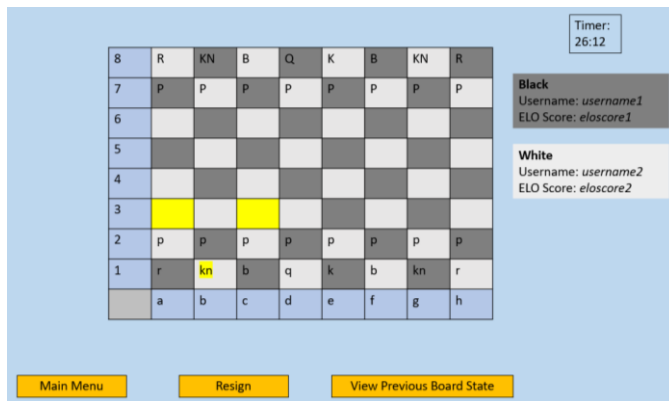
ELO Score: *eloscore2*

8	R	KN	B	Q	K	B	KN	R
7	P	P	P	P	P	P	P	P
6								
5								
4								
3								
2	p	p	p	p	p	p	p	p
1	r	kn	b	q	k	b	kn	r
	a	b	c	d	e	f	g	h

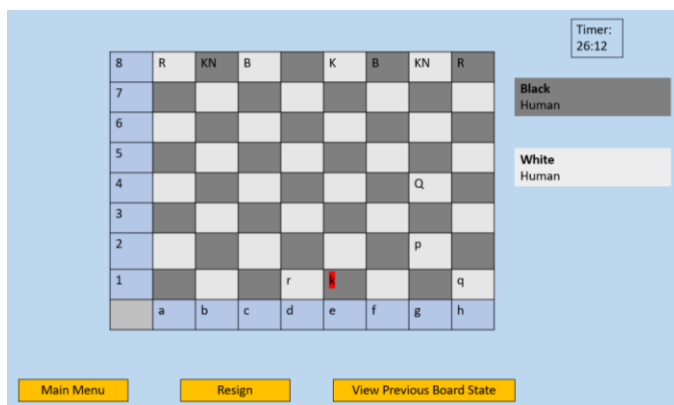
Main Menu

Resign

View Previous Board State



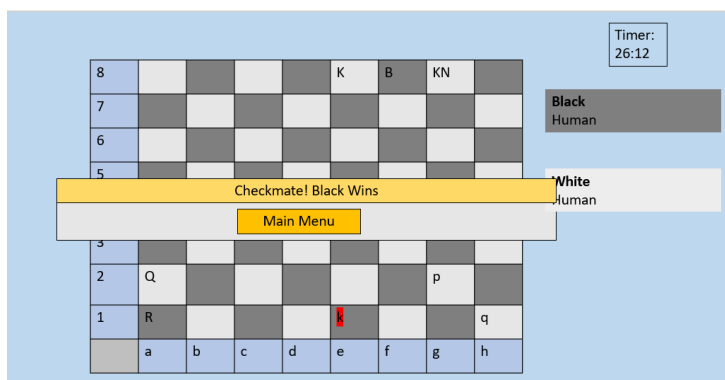
When piece is selected, all of its legal moves are highlighted. This meets my objective of a visually pleasing top down view of a 2d chess board (although the letters will of course be replaced with images of the pieces in the actual system) (objectives 18a and 18b). When a terminal state is reached, a pop-up message is displayed, and the game is deleted by the system once this message is seen. When a piece is in check, the square is highlighted red.



When a pawn, reaches the final rank a message pops-up allowing the user to choose a piece to replace it.



When a terminal state is reached: an appropriate error message is displayed.



The next section of forms will relate to the "Offline Play" section of the system. (Note: some of these are from offline play forms, however, the forms are broadly the same with the only difference being that online forms display usernames and ELO scores).

Offline Play

Slot Number	Opponent	Time Control	Total moves made
1	Person	Unlimited	2
2	Computer – Level 3	40-15	5

Similar to the online, segment, this section allows the user to play up to three games concurrently (objective 4a). To resume a game, one merely has to click on the relevant game slot. The information is featured to allow the user to remember the details of the game and thus select the game they wish to play. The offline play segment allows guests to play games. This is done by locally storing a file of all ongoing guest games. The user also has the option to play against a human or computer opponent.

Play Against Friend

Time Control	15 -10 (Fast)	40-15 (Normal)	Unlimited

When playing against a friend, the user will have the option to customise the time control settings before creating a game. The time controls cater to all types of chess players, with fast being a popular setting amongst skilled players playing for fun. 40-15 is the FIDE standard for tournaments and hence useful for serious chess players. Casual players will prefer to play with an unlimited timer.

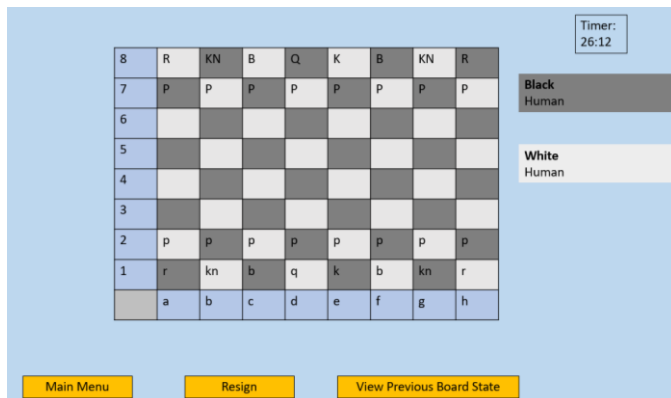
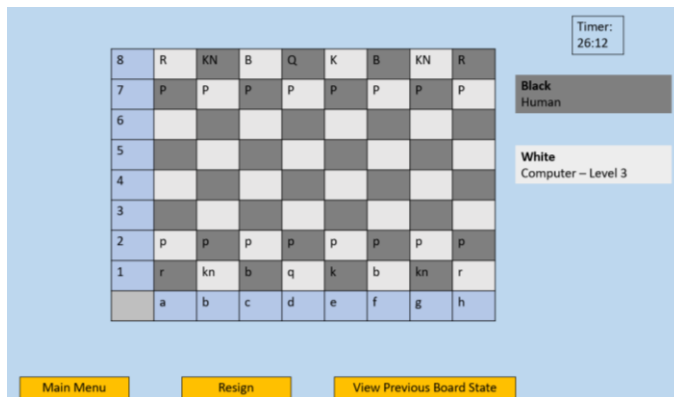
Play Against Computer

Time Control	15 -10 (Fast)	40-15 (Normal)	Unlimited

You play as	White	Random	Black

Computer Difficulty	1 (Easiest)	2	3	4	5 (hardest)

This form allows the user to play against the computer and configure settings such as time control and computer difficulty. There are five different difficulty settings to choose from, something I determined was a good compromise between having too many settings to choose from and too few to choose from. This helps me fulfil objective 5a which requires 5 different computer difficulty settings.



When playing offline, the system displays details about the opponent (such as whether they are a computer or a person).

Data Structures and Methods of Access

In this section, I will discuss all data my system is going to store. Here is the relational database my system will use. Note that this database is in the third normal form.

User (UserID, ELO rating, phone number, password, discord username, discord number, first name, surname, date of birth, Postcode, Address Line 1, Address Line 2, Address Line 3, Address Line 4, ClubName, number of club games won, number of club games drawn, number of club games lost, fixtures, Name of club requested to join)

Note: UserID is equal to username.

Club (ClubName, email address for contact, phone number for contact, leaderID, number of members, start date of current season)

OnGoingGamesOnline (GameID, playerOneID, playerTwoID, boardState, moveNumber, is club game, time control)

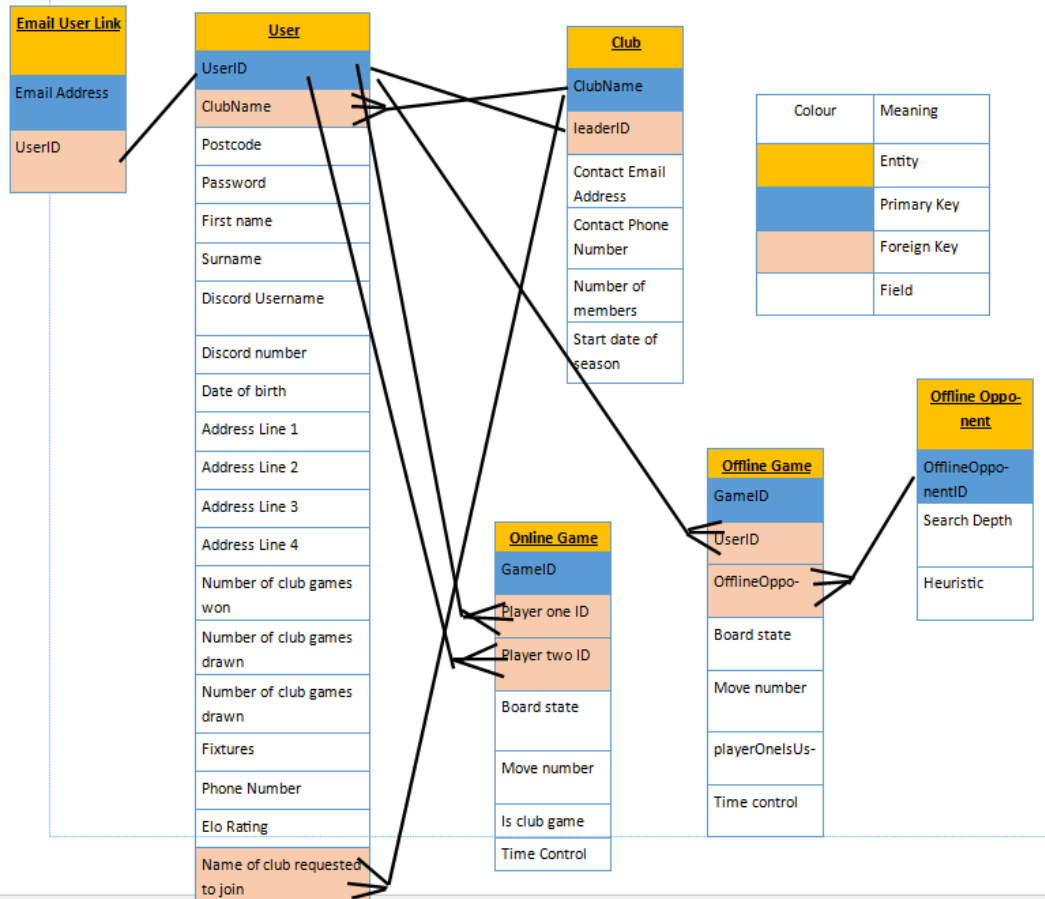
OnGoingGamesOffline (GameID, UserID, boardState, moveNumber, OfflineOpponentID, playerOnesUser, time control)

OfflineOpponent (OfflineOpponentID, searchDepth, heuristic, randomChanceOfMinimaxAlgorithmMakingWrongDecision)

EmailUserLink (emailAddress, userID)

Note that OnGoingGamesOffline and OfflineOpponent will be locally stored data, but the other tables will be stored by the server.

I have also included an entity relationship diagram for my system.



It is also important to highlight that some of these tables will be stored locally by the user, whilst some of these will be stored by the centralised server application. The offline game and offline opponent files will be stored locally by the user, whilst the other files will be stored on the centralised server application.

Data Structure Tables and Validation and Verification

Table Name:	<u>User</u>
Method Of Access:	The file will be accessed sequentially as it will be stored as a text file. However, the effective method of access will be random as the user file will be loaded into an array (a random-access data structure). This array will be used to

	access details quickly using efficient search algorithms such as binary search. The server-side application will have the array storing the user information constantly loaded.			
Field Name	Data Type	Purpose	Length	Example
<u>UserID</u>	String	Used to uniquely identify user. Used when logging in to identify user.	No more than 15 characters	PhilipMortimer6
<u>ClubName</u>	String	Used to store the club that a user is part of. This is done to allow the system to determine which users are club members. Typically, this value will either be "West Cross Chess Club" or left blank. I have chosen to use a string value as opposed to a Boolean value to enable scalability of the system.	No more than 21 characters	West Cross Chess Club
Postcode	String	Used for safeguarding reasons by the club leader.	Between 6 and 8 characters (including both 6 and 8).	SA2 5RT
Password	String	Used to ensure the user account is not compromised and that only the account owner is able to log-in.	More than 5 characters and less than 20.	Ex@mplePas5w0rd23

First Name	String	Used by club leader to identify who each user actually is. A user's name will be displayed in the table if they have filled the field in.	No more than 35 characters	Philip
Surname	String	Used by club leader to identify who each user actually is. A user's name will be displayed in the table if they have filled the field in.	No more than 40 characters	Mortimer
Discord Username	String	Used by the club leader to coordinate voice chat for members.	No more than 20 characters	philipmo47
Discord Number	String	Used by the club leader to coordinate voice chat for members.	Exactly 4 characters	2344
Date of birth	String	Used by the club leader for safeguarding purposes	Exactly 10 characters	03/07/2003
Address Line 1	String	Used by club leader for safeguarding purposes and for coordination of in-person chess sessions.	No more than 40 characters	3 Pennard Avenue
Address Line 2	String	Used by club leader for safeguarding	No more than 40 characters	West Cross

		purposes and for coordination of in-person chess sessions.		
Address Line 3	String	Used by club leader for safeguarding purposes and for coordination of in-person chess sessions.	No more than 40 characters	Swansea
Address Line 4	String	Used by club leader for safeguarding purposes and for coordination of in-person chess sessions.	No more than 40 characters	Carmarthenshire
Number of club games won	Integer	Used by system to calculate the league table of the current season.	1 or 2 characters	5
Number of club games drawn	Integer	Used by system to calculate the league table of the current season.	1 or 2 characters	3
Number of club games lost	Integer	Used by system to calculate the league table of the current season.	1 or 2 characters	2
Fixtures	String	Used to see the fixtures a given user has across a season. The fixture list may be updated if they, or	No limit	JamesL2 LilJ kelly

		another user, leaves or joins the club. The fixture list contains the id of all the users a member still has to play in the current season.		
Phone Number	String	Used to replicate the old system and as a means to contact club members.	No more than 20 characters	01792 401272
ELO rating	Real	Used to give users a sense of their online opponent's skill level	No more than 6 characters (ELO rounded to one decimal place where necessary)	1400.4
Name of club requested to join	String	Used to process requests to join the West Cross Chess Club. If the user has not requested to join a club, or is part of the club, this field will be blank.	No more than 21 characters	West Cross Chess Club

User		
Field Name	Validation / Verification	Pseudocode
UserID	Presence Check	isValid is boolean if userID = "" THEN isValid = FALSE END if

	Length Check – no more than 15 characters	isValid is boolean if lengthOf (userID) > 15 THEN {length of gets length of string} isValid = FALSE END if
	Check that no other user has the same UserID	userFile is array {contains all user records} isValid is boolean if call binarySearch (userFile, userID) NOT= "" THEN {calls a binary search on user file to see if userID exists. Pseudocode for binary search can be found lower down in this document} isValid = FALSE END if
	The userID cannot equal "GuestID39713524" (as this will be used for guests).	isValid is boolean if guestID = "GuestID39713524" THEN isValid = FALSE END if
	UserID can not contain any spaces. (format check).	isValid is boolean for character in userID if character = " " THEN isValid = FALSE END IF Next FOR
Postcode	If field is not left blank, length has to be between 6 and 8 characters (including 6 and 8). (length check).	isValid is boolean postcodeLength is integer set postcodeLength = lengthOf (postcode) if postcodeLength NOT= 0 AND (postcodeLength>8 OR postcodeLength<6) THEN isValid = FALSE END if
	If field is not left blank, it must contain exactly one space. The rest of the field must consist of either capital letters or numbers. (format check).	isValid is boolean noOfSpaces is integer allCapitalLettersAndNumbers is array containsCapitalOrNumber is boolean set containsCapitalOrNumber = false set noOfSpaces = 0 for character in postcode if character = " " THEN noOfSpaces = noOfSpaces +1 ELSE containsCapitalOrNumber = FALSE for element = 1 to lengthOf (allCapitalLettersAndNumbers) if character = allCapitalLettersAndNumbers [element] THEN containsCapitalOrNumber = TRUE END for END if END if Next FOR

		If containsCapitalOrNumber = FALSE THEN isValid = FALSE END if Next FOR if noOfSpaces NOT=0 THEN isValid = FALSE END if
Password	Must be more than 5 characters and less than 20 characters. (length check)	isValid is boolean if lengthOf (passwordOne) < 6 OR lengthOf (passwordOne) > 19 THEN isValid = FALSE END if
	Presence check.	isValid is boolean if passwordOne = "" THEN isValid = FALSE END if
	Double entry verification.	isValid is boolean if passwordOne NOT= passwordTwo THEN isValid = FALSE END if
First name	If the field is not left blank, it must be no more than 35 characters long. (length check).	isValid is boolean if firstName NOT= "" THEN if lengthOf (firstName) > 35 THEN isValid = false END IF END IF
	If the field is not left blank, it must only contain letters. (format check).	isValid is boolean if firstName NOT= "" THEN for character in firstName if character is NOT LETTER THEN isValid = false END if Next FOR END if
Surname	If the field is not left blank, it must be no more than 40 characters long. (length check).	isValid is boolean if surname NOT= "" THEN if lengthOf (surame) > 40 THEN isValid = false END IF END IF
	If the field is not left blank, it must only contain letters. (format check).	isValid is boolean if surname NOT= "" THEN for character in surname if character is NOT LETTER THEN isValid = false END if Next FOR END if

Discord Username	If the field is not left blank, it must contain no more than 20 characters (length check).	isValid is boolean if discordUsername NOT= "" THEN if lengthOf (discordUsername) > 20 THEN isValid = false END IF END IF
Discord number	If the field is not left blank, it must contain exactly 4 characters. (length check).	isValid is boolean if discordNumber NOT= "" THEN if lengthOf (discordNumber) NOT = 4 THEN isValid = false END IF END IF
	If the field is not left blank, it must be an integer. (type check).	isValid is boolean if discordNumber is NOT integer THEN isValid = FALSE END if
Date of birth	If not left blank, it must be in the form dd/mm/yyyy. All non "/" characters must be numbers. (format check)	isValid is boolean {code for "If not left blank" has been demonstrated in earlier fields} elementNo is int set elementNo = 1 for character in dateOfBirth if elementNo = 3 OR elementNo = 6 THEN if character NOT= "/" THEN isValid = FALSE END for END if ELSE If character is NOT integer THEN isValid = FALSE END for END if END IF elementNo = elementNo + 1 Next FOR
	If not left blank, the user's age should not be less than 0 or greater than 150. (range check)	isValid is boolean currentYear is integer age is integer yearOfBirth is integer set yearOfBirth = dateOfBirth[7] + dateOfBirth[8] + dateOfBirth[9] + dateOfBirth[10] age = currentYear – yearOfBirth if age < 0 OR age > 150 THEN isValid = FALSE END if
	If not left blank, some simple validation should be in place to prevent erroneous date of birth inputs. This means that the month input can not be	isValid is boolean dayOfBirth is integer monthOfBirth is integer set dayOfBirth = dateOfBirth[1] + dateOfBirth[2]

	greater than 12, the day can not be greater than 31.	set monthOfBirth = dateOfBirth[4] + dateOfBirth[5] if dayOfBirth > 31 THEN isValid = FALSE END if If monthOfBirth > 12 THEN isValid = false END if
Address Line 1	If not left blank, this field must not be longer than 40 characters (length check).	isValid is boolean if addressLineOne NOT= "" THEN if lengthOf (AddressLineOne) > 40 THEN isValid = FALSE END if END if
Address Line 2	If not left blank, this field must not be longer than 40 characters (length check).	isValid is boolean if addressLineTwo NOT= "" THEN if lengthOf (AddressLineTwo) > 40 THEN isValid = FALSE END if END if
Address Line 3	If not left blank, this field must not be longer than 40 characters (length check).	isValid is boolean if addressLineThree NOT= "" THEN if lengthOf (AddressLineThree) > 40 THEN isValid = FALSE END if END if
Address Line 4	If not left blank, this field must not be longer than 40 characters (length check).	isValid is boolean if addressLineFour NOT= "" THEN if lengthOf (AddressLineFour) > 40 THEN isValid = FALSE END if END if
Phone Number	If not left blank, must contain no more than 20 characters. (length check)	isValid is boolean if phoneNumber NOT= "" THEN if lengthOf (phoneNumber) > 20 THEN isValid = FALSE END if END if
	If not left blank, the field must only contain plusses, spaces and numbers. (format check)	isValid is boolean if phoneNumber NOT= "" THEN for character in phoneNumber if character NOT= "+" AND character NOT = " " AND character is NOT integer THEN isValid = FALSE END IF Next FOR END if

General: user will be asked to confirm all details are accurate before submitting details.

Furthermore, all inputs cannot contain a comma. This is because commas will be used as field separators for all files.

Philip Mortimer

isValid is boolean

for character in input

if character = “,” THEN

isValid = FALSE

END IF

Next FOR

Table Name:	<u>Email User Link</u>			
Method Of Access:	The file will be stored as a text file, which only has sequential access. However, the effective method of access will be random as an array will be loaded which stores the contents of the text file. This array enables for quick searches of the file using algorithms such as binary search (which require random access). This will be very useful when users attempt to log-in to the system using their email address.			
Field Name	Data Type	Purpose	Length	Example
<u>Email Address</u>	String	This is used as an alternative primary key to username when logging in (as many find it easier to memorise their email address).	Less than 40 characters	philip91@aol.com
<u>UserID</u>	String	This is used to link a user's email address to an actual profile. This field is stored to allow users to log-in with their email address or their userID.	No more than 15 characters	LillyJ2

<u>Email User Link</u>		
Field Name	Validation / Verification	Pseudocode
<u>Email Address</u>	Presence check	isValid is boolean if emailAddress = "" THEN isValid = FALSE END if
	Has to be less than 40 characters in length. (length check).	isValid is boolean if lengthOf (emailAddress) > 39 THEN isValid = FALSE END if

	Must contain the @ symbol. (format check)	isValid is boolean containsAt is boolean set containsAt = FALSE for character in emailAddress if character = "@" THEN containsAt = TRUE END FOR END if Next FOR If containsAt = FALSE THEN isValid = false END if
	Email address must be unique, no other user can have the same email address.	emailUserLinkFile is array isValid is boolean if call binarySearch (emailUserLinkFile, emailAddress) NOT= "" THEN isValid = FALSE END if

Table Name:	<u>Club</u>			
Method Of Access:	This file will be accessed sequentially. This is because my current system design will mean that there is only one record stored in this file (for the West Cross Chess Club). As there will be no need to search for a specific club, the file will not require a fast searching algorithm. Hence, the file can be accessed sequentially (as the location of desired data remains the same).			
Field Name	Data Type	Purpose	Length	Example
<u>ClubName</u>	String	This stores the name of the club. At the moment, this will be "West Cross Chess Club". However, to meet my goal of a scalable and maintainable system, I will store the name.	No more than 21 characters	West Cross Chess Club
<u>leaderID</u>	String	This is the userID of the user who is the club leader. This is stored as this person can perform tasks that no other user can, such	No more than 15 characters	JamesL2

		as changing the club's contact details.		
Contact Email Address	String	This is displayed to all users, club members or not. This allows users to contact the club and make enquiries.	Less than 40 characters	jamesl987@outlook.com
Contact Phone Number	String	This is displayed to all users, club member or not. This allows users to contact the club and make enquiries.	No more than 20 characters	01792 431277
Number of members	Integer	Stores the number of members to make fixture scheduling more efficient.	One or two characters	12
Start date of season	String	Used to calculate the end date of the season and to display the date each fixture should be played.	Exactly 10 characters	06/10/2020

Club		
Field Name	Validation / Verification	Pseudocode
Phone Number	If not left blank, must contain no more than 20 characters. (length check)	isValid is boolean if phoneNumber NOT="" THEN if lengthOf (phoneNumber) > 20 THEN isValid = FALSE END if END if
	If not left blank, the field must only contain plusses, spaces and numbers. (format check)	isValid is boolean if phoneNumber NOT="" THEN for character in phoneNumber if character NOT="+" AND character NOT =" " AND character is NOT integer THEN isValid = FALSE END IF Next FOR

		END if
Email Address	Presence check	isValid is boolean if emailAddress = "" THEN isValid = FALSE END if
	Has to be less than 40 characters in length. (length check).	isValid is boolean if lengthOf (emailAddress) > 39 THEN isValid = FALSE END if
	Must contain the @ symbol. (format check)	isValid is boolean containsAt is boolean set containsAt = FALSE for character in emailAddress if character = "@" THEN containsAt = TRUE END FOR END if Next FOR If containsAt = FALSE THEN isValid = false END if
leaderID	Lookup check – userID must be of an already existing user.	isValid is boolean userFile is array if binarySearch (userFile, leaderID) = "" THEN isValid = FALSE END if

Table Name:	<u>Online Game</u>			
Method Of Access:	This file will be stored sequentially as a text file. However, its effective method of access will be random as it will be loaded into an array which is stored whilst the server application is running. The array allows for random data access. Whenever the array is updated, the online text file is updated (to ensure that data is saved). Random access is chosen because when an online game is being played, users need to send updates to the server when making a move. In order for the update to be processed efficiently, the gameId will be searched for using the binary search algorithm and the board state will be updated. Binary search only functions efficiently with random access, hence this file will be accessed randomly.			
Field Name	Data Type	Purpose	Length	Example
<u>GameID</u>	Integer	A numerical value to uniquely represent each ongoing game. The ID is used to efficiently search for and update game records.	Exactly 6 characters	155290

Player One ID	String	Used to determine whether a user is allowed to play in a given game.	No more than 15 characters	pMort5
Player Two ID	String	Used to determine whether a user is allowed to play in a given game.	No more than 15 characters	Lillz4
Board state	String	The current board state is stored to allow both users to play chess and update the board.	68 characters.	<pre> rnbqkbnr 1111 pppppppp oooooooo oooooooo oooooooo oooooooo PPPPPPPP RNBQKBNR </pre>
Move number	integer	Indicates the current move number, used to determine which player's turn it is.	No more than 3 characters	12
Is club game	Boolean	Stores whether a game is a club game. This is used to differentiate between games played in the online segment and games played in the club section.	One character	1
Time control settings	String	Used to allow users to play using different time settings.	No more than 5 characters	40-15

Table Name:	Offline Game
Method Of Access:	<p>This file will be stored as a text file (which has sequential access). The file will be sequentially accessed in order to load it into an array. The array is then used to retrieve data. The array means that offline games are effectively accessed randomly. This is because users can play multiple offline games and hence, games will need to be searched for by their ID's. In order to make the search as quick as possible, the file will need to be (effectively) randomly accessible.</p>

Field Name	Data Type	Purpose	Length	Example
<u>GameID</u>	Integer	A numerical value to uniquely represent each ongoing game. The ID is used to efficiently search games.	Exactly 6 characters	155291
<u>UserID</u>	String	Used to determine which offline games the currently logged in user can access.	No more than 15 characters	pMort5
<u>OfflineOpponentID</u>	String	Used to determine the opponent being played against. This allows the system to use a computer of appropriate difficulty. This also enables friends to play against friends locally.	1 character	2
Board state	String	The current board state is stored to allow users to play chess and update the board.	68 characters.	rn bqkbnr 1111 pppppppp oooooooo oooooooo oooooooo oooooooo PPPPPPPP RNBQKBNR
Move number	integer	Indicates the current move number, used to determine which player's turn it is.	No more than 3 characters	12
playerOnIsUser	Boolean	Used to calculate whether the user goes first, or whether the system moves first.	One character	0

Time control	String	Used to allow for customisable time controls.	No more than 5 characters	40-15
--------------	--------	---	---------------------------	-------

As timer control settings, who player one is, computer difficulty settings and game type will all be dictated by button presses, there is no need for validation as the system will only accept the inputs on these buttons (hence invalid inputs are impossible).

Table Name:	Offline Opponent			
Method Of Access:	<p>The offline opponents will be stored in a sequential text file. However, they will be loaded into an array (a random-access data structure). This allows for random access.</p> <p>This file will be (effectively) accessed randomly. This is because offline opponent IDs will need to be searched for efficiently to enable the fast loading of game records. Random access is needed for efficient searching; hence the file will be accessed randomly.</p>			
Field Name	Data Type	Purpose	Length	Example
<u>OfflineOpponentID</u>	Integer	A numerical value to uniquely represent the offline opponent game. The ID is used to efficiently load offline games. If the ID is 0, it represents the fact that it is a human opponent, not a computer one. All other ID's represent the difficulty of the computer, starting from one (which is the easiest setting).	Exactly 1 character	3
Search Depth	Integer	Stores the depth the search depth of the minimax algorithm. A greater search depth will lead to more nodes being explored	One character	3

		but will be more computationally expensive.		
Heuristic	Integer	Will denote the heuristic used. E.g. heuristic one may indicate a heuristic that counts which user has the most pieces. What each heuristic means will be hard coded into the system	One character	2

Processing Stages

Backup and Recovery

My system will have a comprehensive backup and recovery plan. It will be recommended that the server system is backed up every week. Each week, a copy of all the data files will be saved and stored. There will be three copies of the data at any one time. This method is sometimes referred to as the Grandfather-father-son method. The idea is that each time a new backup is made, the oldest backup is removed completely. So, the son becomes the father, the father becomes the grandfather and the new data becomes the son. The grandfather backup will be stored on an external USB device in order to reduce the likelihood of all data being lost. Additionally, my client has an automatic cloud backup system which backs up all data on his computer to the cloud. Should an error occur with the data, the data will be recovered first by checking for the data in the cloud. If this fails, data will be restored using the son backup. If the son data is corrupted, a backup will be attempted using the father data. If the father data fails, the grandfather data is used. This backup policy is used to reduce the chances of losing or corrupting important user and system data.

My system will make use of two primary types of search.

Binary Search:

```
itemToSearchFor is integer {the value needed to be found}

lower is integer

upper is integer

mid is integer

sortedArray is array {one d array containing integers with the lowest value element being at
element 1 and the highest value element being the last item in the array}

indexOfItemWanted is integer

set indexOfItemWanted = -1 {this is not valid index, hence we know no matches have been found
if this is the final index}

set lower = 1 {lower is the first element of the array}

set upper = lengthOfArray (sortedArray) {upper is the last element of the array}

while lower <= upper

    mid = (upper + lower) DIV 2

    IF sortedArray[mid] = itemToSearchFor THEN

        indexOfItemWanted = mid

        lower = upper + 1

    ELSE IF sortedArray[mid] < itemToSearchFor THEN

        lower = mid + 1

    ELSE

        upper = mid - 1

    END IF

do

{the index of the item wanted has now been found}
```

This code can be updated to function on alphabetical search fields. Using trivial adjustments, this can be used to search 2d arrays which store data. I will use binary search to search user details, using userID as the primary key. If the id is found, the record will be returned. If the id is not found, there will be an empty return (i.e. ""). This will be used when users log-in, as well as when a club leader is appointed (as the system will need to be searched to determine whether a valid ID has been input). For account creation, binary search will be used to see whether a user with that ID already exists. Binary search will also be used when searching for game ID and offline opponent ID records. This will also be used on the email user ID link table to check for valid ID's when logging in.

My system will also make use of linear search. I shall write this for an unsorted array of integers, but like with binary search, this can be applied to 2d arrays of all data types.

Linear Search:

```
itemToSearchFor is integer
unsortedArray is array
indexItemWanted is integer
set indexItemWanted = -1
for element = 1 to lengthOf (unsortedArray)
    if unsortedArray[element] = itemToSearchFor THEN
        indexItemWanted = element
    END if
END for
END IF
Next FOR
```

Linear Search is a very slow search;

however it has to be used on unsorted data. I will use linear search when looking for club members in the user table. I will use linear search as the system is intended to be primarily used by club members. Hence, most of the users will in fact be members of the club. Therefore, linear search is not very inefficient in this regard.

In order for my system to search through data effectively (using binary search), all data needs to be sorted. There are three primary sorting methods which I shall use. They all offer varying trade-offs between speed and memory usage.

Bubble sort:

```
swapMade is boolean
unsortedData is array
buffer is integer
repeat
    swapMade = FALSE
    for element = 1 to (lengthOf (unsortedData) – 1)
        if unsortedData[element] > unsortedData[element+1] THEN
            swapMade = TRUE
            buffer = unsortedData[element+1]
            unsortedData[element+1] = unsortedData[element]
            unsortedData[element]=buffer
        END if
    Next FOR
Until swapMade = false
{unsortedData is now sorted}
```

Insertion sort:

```
unsortedData is array
buffer is integer
indexToBeChecked is integer
set indexToBeChecked = 1
for element = 2 to lengthOf (unsortedData)
    buffer = unsortedData[element]
    for indexToBeChecked = (element - 1) to 0 {indexToBeChecked decrements by one per iteration)
        unsortedData[indexToBeChecked+1] = unsortedData[indexToBeChecked]
    Next for
    unsortedData [indexToBeChecked + 1] = buffer
Next FOR
{unsortedData is now sorted}
```

Merge Sort:

```
declare mergeSort IN data
START
    if lengthOf (data) = 1 THEN
        OUT data
    END if
    middle is integer
    set middle = lengthOf (data) DIV 2
    left is array
    initialise left [middle] {left is an array of length middle with all value initialised}
    right is array
    initialise right [lengthOf (data) – middle]
    for element = 1 to lengthOf (data)
        if element > middle
            right [element-middle] = data [element]
        ELSE
            Left [element] = data[element]
        END if
    Next FOR
    OUT call merge (call mergeSort (left), call mergeSort (right))
END

declare merge IN left, IN right, IN data
START
    leftIndex is integer
    rightIndex is integer
    dataIndex is integer
    set rightIndex = 1
    set leftIndex = 1
```

```
set dataIndex = 1
while leftIndex <= lengthOf (left) AND right <= lengthOf (right)
    if left [leftIndex] < right [rightIndex] THEN
        data [dataIndex] = left [leftIndex]
        dataIndex = dataIndex + 1
        leftIndex = leftIndex + 1
    ELSE
        data [dataIndex] = right [rightIndex]
        dataIndex = dataIndex + 1
        rightIndex = rightIndex + 1
    END if
do
while leftIndex <= lengthOf (left)
    data [dataIndex] = left [leftIndex]
    dataIndex = dataIndex + 1
    leftIndex = leftIndex + 1
do
while rightIndex <= lengthOf (right)
    data [dataIndex] = right [rightIndex]
    dataIndex = dataIndex + 1
    rightIndex = rightIndex + 1
do
OUT data
END
START
sortedData = call mergeSort (unsortedData)
END
```


These sorts all have benefits and drawbacks. Merge sort is extremely fast at sorting large amounts of data but takes up a lot of memory to do so. Insertion sort is very fast with almost sorted data and bubble sort is also fairly quick with nearly sorted data. Therefore, when adding a new user, I shall use insertion sort to sort the user files, as most of the records will already be sorted. As online games may be updated more frequently than user records, I will make use of merge sort for this file. This is also done as the online game file could be rather large (as insertion sort has a significantly inferior time complexity to merge sort) and as many new games could be generated over a short period of time. I do not anticipate using bubble sort, but I may well use it to sort very small data sets. I will also use merge sort to sort offline games and the offline opponent file.

ELO score algorithm:

```
playerOneELO is real
playerTwoElo is real
resultPlayerOne is real {this is 1 if player one wins, 0 if player one loses and 0.5 if it is a draw}
resultPlayeTwo is real {this is 1 if player two wins, 0 if player one loses and 0.5 if it is a draw}
probabilityPlayerOneWin is real
probabilityPlayerTwoWin is real
set probabilityPlayerOneWin = 1 / (1 + 10(playerTwoElo - playerOneElo)/400)
set probabilityPlayerTwoWin = 1 / (1 + 10(playerOneElo - playerTwoElo)/400)
playerOneElo = playerOneElo + 24 * (resultPlayerOne - probabilityPlayerOneWin)
playerTwoElo = playerTwoElo + 24 * (resultPlayerTwo - probabilityPlayerTwoWin)
```

This algorithm will be used at the end of every non-club fixture to update the ELO score of both members. ELO scores start at 1000 for new users. This algorithm will be called every time an online chess game is finished.

League table generation:

```
clubMemberDetails is array {linear search on user details to get and store all club member details}

noOfMembers is integer {stores number of club members}

table is array {2d array with depth equal to number of members and width of 5}

for element = 1 to noOfMembers
    if clubMemberDetails [5] = "" or clubMemberDetails [6] = "" THEN {if either first name or second name is left blank}
        table [1] = clubMemberDetails [1] {name is username}
    ELSE
        table [1] = clubMemberDetails [5] + " " + clubMemberDetails [6]
    END if
    table [2] = clubMemberDetails [14] {number of wins}
    table [3] = clubMemberDetails [15] {number of draws}
    table [4] = clubMemberDetails [16] {number of losses}
    table [5] = table [2] * 3 + table [3] {points}
Next FOR
table = call mergesort (table) {sorts table in order of points (the fifth element of the table array)}
```

This algorithm will be used by the server every time the league table button is pressed, to calculate and display the current league table.

Adding records to a file:

```
recordToAdd is array
file is array
file = LOAD FILE INTO ARRAY {one line = one record, all fields are sperated by commas}
APPEND recordToAdd to file
file = call mergeSort (file)
SET FILE CONTENTS = file
```

I will use this when new online and offline games are created. I will user a similar methodology when adding user records, although I will use an insertion sort instead of a merge sort.

Editing records of a file:

```
updatedRecord is array
elementOfFile is integer
file is array
file = LOAD FILE INTO ARRAY {2d array, each row containing a record and each column a field}
set elementOfFile = call binarySearch (updatedRecord[1], file){returns the element of the record
with primary key updatedRecord [1]}
for across = 1 to widthOf (file)
    file [elementOfFile, across] = updatedRecord[across]
Next FOR
SET FILE CONTENTS = file
```

This will be used when updating records such as online games or when the club leader edits user details.

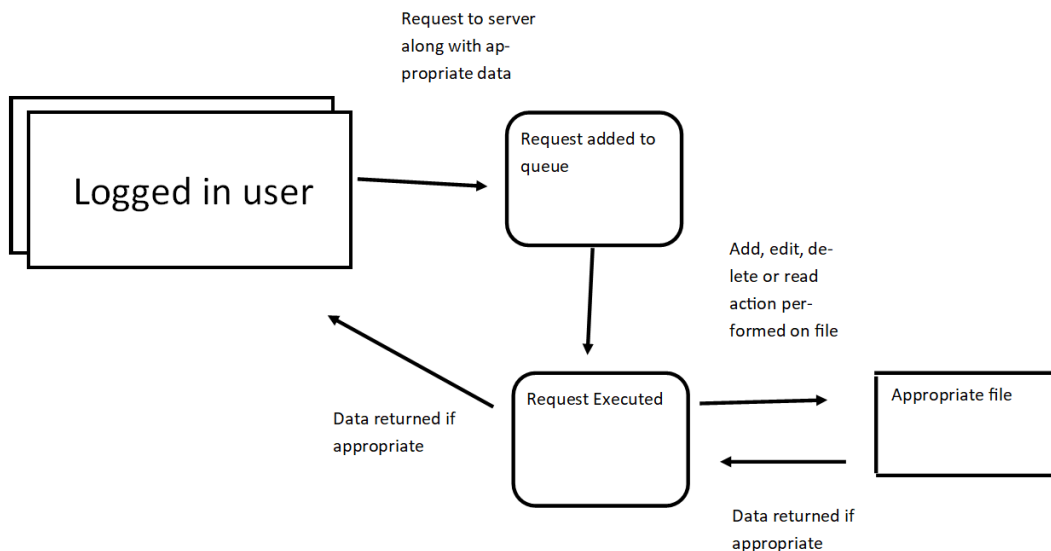
Deleting records in a file:

```
idOfRecordToDelete is integer
file is array
file = LOAD FILE INTO ARRAY
indexOfRecordToDelete is integer
indexOfRecordToDelete = call binarySearch (file, idOfRecordToDelete)
SET FILE CONTENTS = ""
FOR element = 1 to indexOfRecordToDelete-1
    WRITE file [element] TO FILE ON NEW LINE
Next FOR
For index = indexOfRecordToDelete +1 to lengthOf (file)
    WRITE file[index] TO FILE ON NEW LINE
Next FOR
```

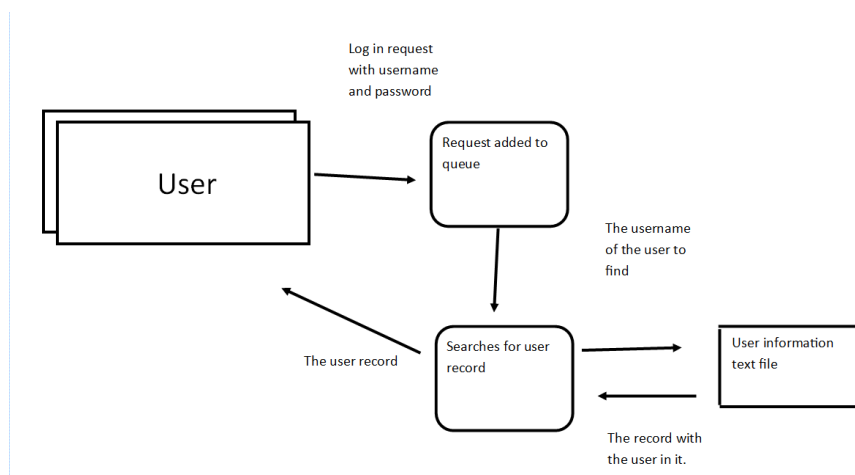
This will be used by the server application to remove games that have finished from the ongoing online games file. This will also be used by the system to delete finished offline games.

For my server application, I will make use of a data structure known as a queue to store and process all requests made to the server application. These requests will essentially boil down to the add, editing and deleting of data. If a user wants to create a new account, this will be sent to the queue which will add a new record to the user and email user link files. When a new online or offline game, is created, records are added to the appropriate files. When changing a club leader, or user details,

or adding a user to the club or updating a board state, appropriate files are edited. If an online or offline game is finished, the appropriate game record is deleted. My server application will use this queue structure to add to, edit and delete from the appropriate files. A queue is a structure that processes the oldest request at any given time.



For example, when logging in:



The user would request their user record based on the log-in credentials provided. The server returns the appropriate user record or an error message. Once the user gets their user record, the system checks to see if the password matches the inputted password. The system also checks to see if the user is a club member or club leader. If the password is correct, the log-in is successful. The user access level is assigned based on whether the user is a member, leader or neither. Players logging in as a guest are given the access level of a guest.

User Access Level Calculation

{Note guests access level is assigned by having the user press a button titled “Log in as Guest”}
 userRecord is Array
 isClubMember is boolean
 isClubLeader is boolean
 clubLeaderId is String
 set isClubMember = FALSE

```
set isClubLeader = FALSE
set userRecord = getUserRecordFromServer (userName) {gets user record from server}
set clubLeaderId = getLeaderIdFromServer() {gets the username of the club leader from the server}
if userRecord [0] = "userRecordNotFound" {checks to see if username was found} THEN
    OUTPUT "User not found"
    return
if userRecord [3] NOT = password THEN
    OUTPUT "Incorrect Password Entered"
    Return {exits method}
END if
If userRecord [2] = "West Cross Chess Club" THEN {checks to see if user is a club member}
    isClubMember = TRUE
END if
If userRecord [0] = clubLeaderId {checks to see if user is the club leader} THEN
    isClubLeader = TRUE
END if
```

Fixture Generation:

```
declare generateFixtures IN names
START
    noOpponentId is String
    set noOpponentId = "NO_OPPONENT"
    If lengthOf (names) = 0 THEN
        Return EMPTY ARRAY {returns two d array that is empty}
    END if
    if lengthOf (names) MOD 2 = 1 THEN
        newName[lengthOf(names) +1] is array
        for element = 1 to lengthOf(names)
            newName[element]=names[element]
        Next FOR
        newName[lengthOf(names)+1] = noOpponentId
        names [lengthOf (newNames)] is array
        for element = 1 to lengthOf (names)
            names [element] = newName [element]
        Next For
    END if
    buffer is String
    swap is integer
    for element = 1 to lengthOf (names)
        swap = GET RANDOM NUMBER BETWEEN 1 AND lengthOf (names) {includes both
1 and length}
        buffer = names [element]
        names [element] = names [swap]
        names [swap] = buffer
    Next for
    index is integer
    fixtures [lengthOf (names)] [2*lengthOf(names) -2] is array
```

```

    for y = 1 to lengthOf(names)
        for x = 1 to lengthOf(names) - 1
            index = yVal + xVal
            if index >= lengthOf(names) THEN
                index = index - lengthOf(names) + 1
            END if
            fixtures [y,x] = names [index]
        NEXT for
    NEXT for
    xVal is integer
    for y = 1 to lengthOf (names)
        xVal = lengthOf(names)
        for x = 1 to lengthOf(names) - 1
            fixtures [y, xVal] = fixtures [y,x]
            xVal = xVal + 1
        NEXT for
    NEXT for
    OUT fixtures
END

```

This algorithm will be used to generate all club fixtures, so that all club members play each other exactly twice across a season.

For the actual game of chess itself, I will use object-oriented programming to create a game object. This game object will store all needed information such as board state, previous board state, player details, rule set and current time. This object will be used to represent the board after receiving a game record. This object will make use of methods such as `getAllAvailableMoves` to retrieve all possible moves from a given piece. This will be used to highlight all available moves when a user clicks on a piece. This will require implementation of simple chess logic (such as Bishop's being only able to move diagonally or knights being able to move two squares in one axis and one in the other, hopping over pieces as they do this). I will also have methods that retrieve all legal moves at any time, something which will be useful in the implementation of the computer algorithm. In order to add a move, I will use a method called `addMove` which makes use of the piece's current and future coordinates to update the board state. I will also have a method which allows the user to go back to a previous board state, something useful when trying to see what move the opponent has played, and also something useful for implementation of the minimax algorithm. The board will be represented using single characters of text e.g. "P" represents a black pawn, whilst "p" represents a white pawn. "0" represents an empty square.

Alpha-beta pruned, depth-limited minimax algorithm:

```

declare min IN movesAvailable, alpha, beta, currentDepth, depthLimit
START
    minVal is integer
    set minVal = 3
    indexOfBest is integer
    isTerminalNode is boolean
    bestMove[5] is array {stores the x,y cords of the move before and after and the value of
the move}
    for element 1 to lengthOf (movesAvailable)

```

```

        call addMove(movesAvailable[element,2],movesAvailable[element,3],
movesAvailable[element,4]), movesAvailable[element,5])){adds one of available moves from
previous board state)
        isTerminalNode = call isGameOver
        if currentDepth = depthLimit OR isTerminalNode = TRUE THEN
            movesAvailable [element,1] = call getValueOfBoardUsingHeuristic {stores
value of board after move with heuristic}
        ELSE
            nextMoves is array
            nextMoves = call getAllPossibleMoves
            movesAvailable [element,1] = call max (nextMoves, alpha, beta,
currentDepth + 1, depthLimit)
        END IF
        If movesAvailable [element,1] < minVal THEN
            minVal = movesAvailable [element,1]
            indexOfBest = element
        END if
        call undoMove {undos move added, i.e. reverts board state}
        if minVal <= alpha THEN
            best [1] = minVal
            best [2] = movesAvailable [indexOfBest,2]
            best [3] = movesAvailable [indexOfBest, 3]
            best [4] =movesAvailable [indexOfBest,4]
            best [5] = movesAvailable [indexOfBest, 5]
            OUT best
        END IF
        If minVal < beta THEN
            beta = minVal
        END IF
    NEXT for
    best [1] = minVal
    best [2] = movesAvailable [indexOfBest,2]
    best [3] = movesAvailable [indexOfBest,3]
    best [4] =movesAvailable [indexOfBest,4]
    best [5] = movesAvailable [indexOfBest, 5]
    OUT best
END
declare max IN movesAvailable, alpha, beta, currentDepth, depthLimit
START
    maxVal is integer
    set maxVal = -3
    indexOfBest is integer
    isTerminalNode is boolean
    bestMove[5] is array {stores the x,y cords of the move before and after and the value of
the move}
    for element 1 to lengthOf (movesAvailable)
        call addMove(movesAvailable[element,2],movesAvailable[element,3],
movesAvailable[element,4]), movesAvailable[element,5])){adds one of available moves from
previous board state)
        isTerminalNode = call isGameOver
        if currentDepth = depthLimit OR isTerminalNode = TRUE THEN

```

```

        movesAvailable [element,1] = call getValueOfBoardUsingHeuristic {stores
value of board after move with heuristic}
    ELSE
        nextMoves is array
        nextMoves = call getAllPossibleMoves
        movesAvailable [element,1] = call min (nextMoves, alpha, beta,
currentDepth + 1, depthLimit)
    END IF
    If movesAvailable [element,1] > maxVal THEN
        maxVal = movesAvailable [element,1]
        indexOfBest = element
    END if
    call undoMove {undos move added, i.e. reverts board state}
    if maxVal >= beta THEN
        best [1] = maxVal
        best [2] = movesAvailable [indexOfBest,2]
        best [3] = movesAvailable [indexOfBest, 3]
        best [4] =movesAvailable [indexOfBest,4]
        best [5] = movesAvailable [indexOfBest, 5]
        OUT best
    END IF
    If maxVal < beta THEN
        alpha = maxVal
    END IF
NEXT for
best [1] = maxVal
best [2] = movesAvailable [indexOfBest,2]
best [3] = movesAvailable [indexOfBest,3]
best [4] =movesAvailable [indexOfBest,4]
best [5] = movesAvailable [indexOfBest, 5]
OUT best
END
declare computer move
START
    result is array
    if moveNumer MOD 2 = 0 THEN
        result = call min (call getAllPossibleMoves, -3, 3, 1 , depthLimit)
    ELSE
        Result = call max (call getAllPossibleMoves, -3, 3, 1 , depthLimit)
    END IF
    call addMove (result [2], result [3], result [4], result [5])
END

```

This algorithm looks at all possible board states up to a given depth and evaluates the value of these states using a heuristic. The board state has a value of 1 for a win, 0 for a draw and -1 for a loss. If a node is not a terminal board state but at the depth limit, the heuristic will return some value to attempt to approximate this. E.g. 0.67 would indicate a high probability of a win. I will use a range heuristic. My most basic heuristic will be calculating value by adding by the pieces of both sides with relative multipliers and compare which side is in the most advantageous position. I shall use the

heuristic that a pawn is worth one point, a queen is worth 9 points, a rook is worth 5 points and both knights and bishops are worth 3 points. Of course, this value will then be normalised to be in the range of -1 to 1. More advanced opponents will combine greater search depths with more advanced heuristics. I shall make use of piece-square tables which determine the value of a piece depending on its position. The tables I shall use are outlined here

https://www.chessprogramming.org/Simplified_Evaluation_Function

By combining these functions, along with a slightly fine-tuned piece valuation system proposed by Chess grandmaster Gary Kasparov (which proposes the following values:

Pawn	Knight	Bishop	Rook	Queen
1	3	3.15	4.5	9

), I will achieve five different computer difficulty settings.

By making use of my design, I will be able to create an effective system that fulfils all of my objectives and all of my client's requirements.