JAR Chess Game Use Case Survey

# Actor Summaries

## Player

Players will use the game to play matches and access saved historical matches.

* Speed - The application needs to be responsive or players will find other games to play.
* Ease of use - The UI needs to be intuitive. All components of the application need to be easily understood and managed without training or external documentation. Most players won’t bother putting that much effort to learn to use the app.
* Security - While there is not much for sensitive data, the app needs to protect players’ username\password combinations and email addresses.
* Privacy – Players don’t want their information in the hands of anyone they didn’t give it to, and they won’t want us monitoring their location or other private data without cause and consent.
* Availability – If any aspect of the game is not available at any given time, that will make the players look elsewhere for entertainment.

### Logged-Out Player

A player that is not logged in. This player will be able to have some in game functionality but will not be able to play in online matches, access a friends-list, or perform other actions that require a player be logged-in. They will be able to play local games.

### Logged-In Player

A player that is currently logged in. A logged-in player will be able to access all features of a player apart from logging in, verifying email, recovering account, and creating accounts.

## Friend (secondary actor)

A friend of the player that may or may not be a player themselves.

## Opponent (secondary actor)

The opponent is entity that the player is playing against.

# Use Case Summaries

## Mid-Match Use Case Summaries

### Move Piece

The goal of this use case is to allow a player to move a piece under their control. The player tells the system that they want to move a particular piece they control to a particular coordinate[[1]](#footnote-2). The system checks to see if the move is allowed. If the move is allowed, the system executes the move. The system then checks to see if the opponent is in check. If the opponent is in check, the system checks to see if it is a checkmate. If it is a checkmate, the system designates the player as the winner of the match. If opponent is in check but not checkmate, the system tells the player and opponent that the opponent is in check. If the move involves a pawn promotion, the use case is extended with the [Promote Pawn](#_Promote_Pawn) use case. Then that the system toggles the player clock, and lets the opponent know that it is their move.

### Promote Pawn

The goal of this use case is to allow a player to promote a pawn to a rook, bishop, knight, or queen of the same color. This use case extends the [Move Piece](#_Move_Piece) use case. The system asks the player to choose what the pawn will be promoted to. The player makes a choice, and then the system will promote the piece.

### Request Pause

The goal of this use case is to allow a player to request a match be paused. The player tells the system they would like to pause the match. The system asks the opponent if they will accept the request. The opponent tells the system their answer. If the opponent agreed, the system will pause the game until both players tell the system that they are ready to resume. If the opponent did not agree, the system will let the player know that the request was rejected.

### Resign from Match

The goal of this use case is to allow a player to resign from a match. The player tells the system that they want to resign from the match. The system tells the player and opponent that the opponent is the winner of the match, and offers to save the match as a suspended match before the resignation, or a historical match (see the [Suspend Ongoing Match](#_Suspend_Ongoing_Match) and [Save Historical Match](#_Save_Historical_Match) use cases). The player indicates all desired ways to save if any. The system extends with the chosen use cases.

## After-Match Use Case Summaries

### Suspend Ongoing Match

The goal of this use case is to allow the player to suspend an ongoing game in order to allow it to be resumed at a later time. When the player or the opponent resign from a match, the system asks the player if they would like to suspend the game. The player tells the system their answer. If the player said yes, the system suspends the game so that it may be resumed from the point just before the resignation.

### Save Historical Match

The goal of this use case is to allow the player to save a completed match locally for review and sharing. When a match is over the system askes the player if they want to save the match as a historical match. The player tells the system their answer. If the player tells the system yes, the system will save the game.

## Account Use Case Summaries

### Create Account

The goal of this use case is to allow the logged-out player to create an account for online play. The player tells the system that they want to create an account. The system asks the player for a username, email, and desired password. The player provides a username, email address, and desired password. If nothing is wrong with the provided information[[2]](#footnote-3), the system checks to see if the username or email are already associated with an existing account. If the email and username are not already used[[3]](#footnote-4), the system asks the player to verify the password. The player verifies the password, and then the system creates a nonverified account and sends a verification email (see the [Verify Email](#_Verify_Email) use case).

### Login to Account

The goal of this use case is to allow the logged-out player to login to an account. The player tells the system they wish to log in. The system asks for the player’s login credentials. The player provides the login credentials. The system verifies that the credentials are linked to an account[[4]](#footnote-5).If the credentials match an account, the system checks to see if the account has had its email verified. If email verification is needed, the system will request a verification code to proceed *(see* [Verify Email](#_Verify_Email)*use case).* If/once the account’s email has been verified the system will log the player in. If the player logged in with a temporary password from an account recovery (see the [Recover Account](#_Recover_Account) use case), the system will ask the player to provide a new desired password (see the [Change Password](#_Change_Password) use case).

### Verify Email

The goal of this use case is to allow the logged-out player to verify the email address used when creating an account. The player tries to log in (see the [Login to Account](#_Login_to_Account) use case) with a nonverified account. The system notifies the player that the email account has not been verified and asks the player to enter the verification code that was sent to the email account. The player checks for the verification email. If the verification email is found[[5]](#footnote-6), they enter the provided code. If the code matched the expected value[[6]](#footnote-7), the system sets the account as verified and allows the player to login as normal, see the [Login to Account](#_Login_to_Account) use case.

### Recover Account

The goal of this use case is to allow the logged-out player to recover their account in the event that they forgot their login credentials. The player tells the system that they need to recover their account. The system asks for an email address associated with the account. The player provides the email address. If the system finds the email address linked to an account, it sends a temporary password to that email. The system tells the player that, if an account was linked to the provided email address, a temporary password was sent to that email address and instructs the player to use that temporary password to login (see the [Login to Account](#_Login_to_Account) use case).

### Change Password

The goal of this use case is to allow a logged-in player to change their password. This use case has a prerequisite that the player must me logged in. The player tells the system that they want to change their password. The system askes the player to enter their current password. The player enters their current password. The system asks the player to enter the desired password. If the password is a valid password[[7]](#footnote-8), the system will ask the player to verify the desired password. The player will verify the desired password and then the system will save the new password for the player’s account.

### Logout of Account

The goal of this use case is to allow the logged-in player to logout of an account. A prerequisite for this use case is that the player must be logged into an account. The player tells the system that they want to logout and the system logs the player out.

## Customization Use Case Summaries

### Change Avatar

The goal of this use case is to allow the player to be able to change their in-game avatar. The player will let the system know that they wish to change their avatar. The system then shows the player a selection[[8]](#footnote-9) of options to choose from. The player than chooses what avatar they want to use. The system saves this and uses that avatar for the player from then on out until the player goes through this process again.

### Change Gameboard and Piece Appearance

The goal of this use case is to allow a player to change the appearance of their in-game pieces and board. The player will let the system know that they wish to change the appearance of the board and pieces. The system will offer a selection[[9]](#footnote-10) of options to the player and await the player’s choice. The player selects an option. The system saves the change and uses the selected appearance from then on out until the player goes through this process again.

## Match Making Use Case Summaries

### Create Single Player Match

The goal of this use case is to allow a player to create a single player match. The player lets the system know that they want to create a single player match. The system asks the player what difficulty they want. The player tells the system what difficulty they want. The system asks the player to make any desired match setting changes. The player changes setting values from defaults for anything they want to change and tells the system when they have made all changes. The system starts a match against an AI opponent at the desired difficulty.

### Create Local Multiplayer Match

The goal of this use case is to allow a player to create a local match. The player lets the system know that they want to create a local multiplayer match. The system asks the player to choose desired settings for the match. The player changes setting values from defaults for anything they want to change and tells the system when they have made all changes. The system starts a match against a local opponent with the desired settings.

### Find Online Match with Matchmaking

The goal of this use case is to allow the logged-in player to get put into a match with another player chosen by the system’s matchmaking process. The player tells the system they want to find an online match with matchmaking. The system find’s an opponent from other players looking for matches with matchmaking. The system starts a match with the two players.

### Create Custom Online Match

The goal of this use case is to allow the logged-in player to create a private online match and invite a friend to play. The player lets the system know that they want to create a private online match. The system asks the player to tell it what custom settings are desired for the match[[10]](#footnote-11). The player changes desired settings from default values and lets the system know when all changes are made. The system asks the player to indicate a friend to invite. The player selects a friend to invite. The system notifies the invited friend (see the [Join a Friend’s Online Match](#_Join_a_Friend’s) use case in the Notifications Use Case Summaries). When the friend joins, the system starts the match.

## Suspended Match Use Case Summaries

### View Suspended Matches

The goal of this use case is to allow the player to view all the currently suspended matches. The player tells the system that they want to view suspended matches. The system shows the player a list of all suspended games that the player can access.

### Resume Suspended Match

The goal of this use case is to allow the player to resume an ongoing game that was saved. The player tells the system that they want to resume an ongoing match. The system asks the player to indicate the desired opponent[[11]](#footnote-12). The player tells the system their choice. The system resumes the match.

### Remove Suspended Match

The goal of this use case is to allow the player to remove an ongoing game that was saved. A prerequisite of this use case is that a game must have been suspended by the player that has not already been removed. The player lets the system know that they want to remove a suspended game. The system asks the player if they are certain that they wish to delete the suspended match. The player tells the system yes or no. If the player says yes, the system removes the suspended game.

## Historical Match Use Case Summaries

### View Historical Match

The goal of this use case is to allow the player to view a step by step replay of a saved match. The player tells the system they want to view a historical match. The system presents the player with a list of available historical matches if any exist. The player chooses a match to view from the options presented. The system shows the starting state of that match and offers a navigation bar to allow the player to see each move in the match step by step.

### Share Historical Match

The goal of this use case is to allow the player to share a historical match with other people. The prerequisite for this use case is that the player must be viewing a historical match (see the [View Historical Match](#_View_Historical_Match) use case). The player tells the system that they want to share the match. The system asks the player to indicate a method of sharing. The player tells the system how they want to share[[12]](#footnote-13). The system asks the player to select from a list of available formats if more than one exists for the method of sharing selected. The player picks one format. The system converts the match as needed and outputs the match according to the selected method of sharing.

### Remove Historical Match

The goal of this use case is to allow the player to remove a saved historical match. The prerequisite for this use case is that the player must be viewing a historical match (see the [View Historical Match](#_View_Historical_Match) use case). The player tells the system they want to remove a historical match. The system asks for confirmation that the player wishes to remove the historical match. The player confirms the choice. The system removes the historical match.

## Friends List Use Case Summaries

### Add Friend

The goal of this use case is to allow a player to add a friend to their friends-list. Prerequisites to this use case are that the player must be logged in, the friend must have an account, and that the player must know the username of the friend. The player tells the system that they want to add a friend. The system asks for the username of the friend. The player provides the username of the friend. The system checks for an account with the provided username and gives the friend a notification about the friend invite (see the [Accept Friendship Request](#_Accept_Friendship_Request) use case).

### Remove Friend

The goal of this use case is to allow a player to remove a friend from their friends-list. Prerequisites to this use case are that the player must be logged in and the player must have a friend on their friends list. The player tells the system that they want to remove a friend. The system removes the player and friend from each other’s friends-lists.

## Notification Use Case Summaries

### View Notifications

The goal of this use case is to allow the player to view notifications. If there are unseen notifications, the system shows the player a symbol to notify them that there are unseen notifications. The player tells the system they want to view notifications. The system shows all notifications to the player, with an indicator[[13]](#footnote-14) letting the player know which notifications are new if any. The player selects a notification to view. The system shows the player the full details of the selected notification.

### Accept Friendship Request

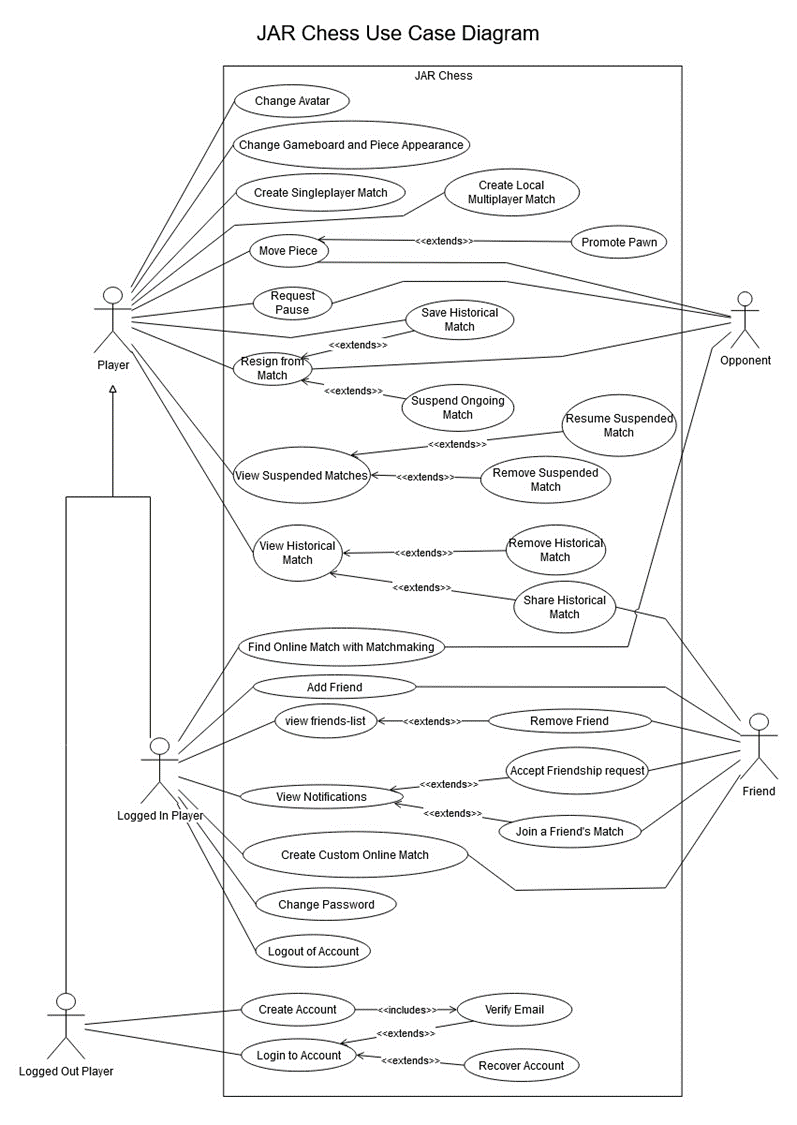
The goal of this use case is to allow a player to accept a friendship request. Friendship requests appear to players as notifications. The player tells the system they want to accept a friendship request. The system adds the friend and the player to each other’s friends-lists.

### Join a Friend’s Match

The goal of this use case is to allow the logged-in player to join a game created by a friend. A prerequisite for this use case is that a friend must have created a private online match (see the [Create Custom Online Match](#_Create_Custom_Online) use case) and invited the player. The system had already added an invite to the player’s notifications as part of the invitation process. The player views notifications (see the [View Notifications](#_View_Notifications) use case), notices the invite, and tells the system that they accept the invite. The system will connect the player to the match and will start the match.

(JAR Chess Use Case Diagram on next page)

# Use Case Diagrams



1. This exact way this move declaration is made depends on preference settings. For example, some players may want to tap a piece and tap a location, others may wish to slide the piece. Requiring a confirmation before executing the move is another option that may be enabled or disabled. [↑](#footnote-ref-2)
2. If there is a problem with any of the information the system tells the player and asks for a different value. [↑](#footnote-ref-3)
3. If either the email or username is in use, the system tells the player what is not available and asks for a different value. [↑](#footnote-ref-4)
4. If the entered information is not linked to an account, the system tells the player that the credentials don’t match any accounts and asks them to try again or recover their account (see the [Recover Account](#_Recover_Account) use case). [↑](#footnote-ref-5)
5. If a verification email is not found, the player asks the system to resend a new verification code, and the system sends an email with a new code and waits for the user to provide the code. [↑](#footnote-ref-6)
6. If the code doesn’t match the expected value, the system tells the player that the code was not accepted and waits for the next code to be entered. [↑](#footnote-ref-7)
7. Passwords must meet a set of conditions intended to promote the use of strong passwords. [↑](#footnote-ref-8)
8. There will be an initial selection of basic avatars. As you win matches, a greater selection will become available. There can also be certain avatars achieved if certain milestones are achieved. For example, win five consecutive games. [↑](#footnote-ref-9)
9. The piece and board appearance options can be progressively unlocked during play. As you reach certain scores or milestones, more options will become available. [↑](#footnote-ref-10)
10. These will include speed-play time limits, move limits, and other similar options for the custom match. [↑](#footnote-ref-11)
11. Opponents may be AI, local player, or friend. [↑](#footnote-ref-12)
12. Methods of sharing may include sharing with friends within the JAR Chess community, sending an email, or sending a text message. To share with friends within the JAR Chess community the player would need to be logged in. [↑](#footnote-ref-13)
13. Perhaps a different color background, a bright boarder around the item, or a logo of some sort in front of the message. [↑](#footnote-ref-14)