

Bravo Design Review

Game Logic Team

Department of Computer Science, University of Calgary

Seng300 Introduction to Software Engineering

Steve Sutcliffe

March, 21, 2025

Summary

Our review of the project files and diagrams identified several key areas for improvement and organization.

- **Completeness of Game Diagrams:** While **Chess** and **Connect4** have full sets of game diagrams, **Go** and **TicTacToe** do not. TicTacToe only includes a **class diagram and a use case description**, while Go has only a **use case description and a diagram**.
- **Consistency & Organization:** It would improve clarity if diagrams were compiled together and if related components (such as class diagrams, use case descriptions, and use case diagrams) used **consistent naming conventions** to make cross-referencing easier.
- **Game Logic Diagrams:** A more structured approach could be beneficial—consider organizing them into **one overarching diagram** where each game extends shared **abstract classes** (similar to Assignment 2). This would reflect shared mechanics like **Player, Board,**

etc. across all three games.

- **Integration Class Diagram Issues:** The **integration class diagram** is unclear, referencing **nonexistent functions** and **lacking connections** to subfunction classes. It's uncertain if different sections connect properly (e.g., **profile may need to be linked to authentication**).
- **Use Case Diagrams:** These are well-structured and **effectively describe expected system behavior**.
- **Class Diagram Challenges:**
 - Found to be **less clear** overall, with significant **variation in quality and structure**.
 - **More explicit connections** between different classes are needed to avoid **floating, unlinked classes**. Even placeholder ("stand-in") classes for external references would improve cohesion.
 - Some diagrams are **overcomplicated**, particularly in cases where **interfaces are only used by a single class**.
- **System Gaps & Missing Links:**
 - Some components **are not designed to work together**, causing **gaps in system functionality**.
 - The **game logic functions referenced in the integration diagram are missing from game logic diagrams**.
 - Since similar functions exist in subclass diagrams, **all functions referenced in integration should also appear in subclass diagrams** to ensure consistency.

Conclusion







While the use case diagrams are effective, improvements in **consistency, organization, and integration** are needed for class and game logic diagrams. Establishing **better connections, reducing overcomplication, and ensuring function consistency** will enhance clarity and system coherence.

GUI Elements (Pictures)

General comments:

- When a player signs up with an account, consider asking for their email for extra verification purposes instead of just a username.
- Consistent colour scheme and clean look

Annotated files:

-  ANNOTATED_Sign-Up_Page.png
-  ANNOTATED_Log-In_Page.png
-  ANNOTATED_Landing_Page.png
-  ANNOTATED_Dashboard.png
-  Start page critique .png
-  end_game. critique.png

GUI

Class diagram seems to be missing update functions for the game display

GUI use case diagram

Pros:

1. Comprehensive Functionality Representation
 - The diagram effectively includes core features such as:
 - Login/Registration
 - Player & Game Search
 - Matchmaking & Leaderboards
 - Friends List & Player List
 - Online Game Sessions & In-game Chat
 - Game Availability & History Tracking
 - Logout & Exit System
 - This ensures a well-rounded representation of all major interactions a player might need.
2. Well-Defined External Interactions
 - The system clearly interacts with external components like:
 - Player Database for storing player data.
 - Game Server for managing game sessions.
 - Messaging System for in-game chatting.
 - These additions help define the role of external systems within the architecture.
2. Separation of Concerns
 - The diagram maintains a clear separation between GUI elements and backend systems like the Player Database, Game Server, and Messaging System.
 - This ensures a logical structure that distinguishes between front-end user interactions and back-end functionalities.

Cons:

1. Cluttered Layout with Many Overlapping Lines
 - The diagram is somewhat dense, with several dashed lines crossing each other, making it difficult to follow the relationships between use cases.
 - This could reduce readability, especially for someone new to the system.
2. Lack of User Actions Beyond the Dashboard
 - The Dashboard is the primary hub for navigation, but some user actions outside of it (like in-game interactions) are not explicitly detailed.
 - For example, how does a player interact with the game once they join an online session?
 - The flow after game initiation could be better detailed.

3. No Explicit Error Handling or Connection Issues

- The system does not seem to account for:
 - Login failures
 - Server connection loss
 - Failed matchmaking or unavailable games
- Including error-handling use cases would improve system robustness.

Suggestions:

1. Improve Diagram Readability

- Rearrange elements to reduce crossing lines and overlapping dependencies.
- Consider grouping related functions to make dependencies clearer.

2. Clarify Some 'Extend' Relationships


- Ensure all extended use cases clearly indicate when and why they are triggered.
- Consider merging redundant use cases if they don't need to be optional.

3. Expand User Actions Beyond the Dashboard

- Add details about in-game interactions (e.g., making a move in a game, game-specific actions).
- Clarify how players return to the dashboard after a match.

4. Incorporate Error Handling Use Cases

- Introduce cases for:
 - Login failures
 - Connection loss
 - Failed game join attempts
 - Matchmaking timeouts
- This would improve the realism of the system's behavior.

 annotated_use_case_description_GUI.pdf

 seng300_class_diagrams_gui_critique.png

Authentication

Class Diagram

Pros:

- Various arrows were used for clarity
- Functions in the classes all had clear purpose

Cons:

- Scenario of a class inheriting another where it is unnecessary to do so
- Partially clear how this diagram relates with other classes with integration class diagram, however more clarity on which functions connect and how they connect will result in a clearer and more effective class diagram

The system could be enhanced with creating temporary match histories so guest users can create a user profile after the game and have their matches saved, and by adding a removeAllHistory function so if users want to delete their entire history they will not have to delete each match one by one.

Annotated class diagram is shown below:

[Annotated Authentication Class Diagram](#)

Use Case Description and Diagram

Pros:

- Covered all main use cases in diagram
- Added developers and testers use cases to create a system that allows for continuous development
- Arrow specification to allow for a clear diagram
- Clear explanations of each case

Cons:

- Small unnecessary connection in use case diagram
- Bot overloading being brought up as a concern but no use cases created to protect against it
- Match history being noted as optional, but clarity on whether it is optional by guest users not having it, or by signed in users being able to turn it off is unclear
- The extend of player deletion is not clarified, as match history may contain player usernames

Annotated use case diagram and description is shown below:

[Annotated Authentication Use Case Diagram and Description](#)

Matchmaking


Matchmaking Class Diagram

Pros:

- Clear actor roles. This helps in understanding the flow of matchmaking
- Well defined use case structure.

Cons:

- Some of the relationships between use cases are unclear, especially when and how these processes are triggered. Further explanation would help clarify this.
- Error handling is limited. Adding considerations for scenarios like failed matchmaking or invalid data would improve the system.
- There's no "Cancel Matching" use case. Adding it would enhance the user experience and satisfaction.
- The "Match Found" process only connects to the opponent. To ensure a complete experience, both players should be notified.

Suggestion for improvement and clarification in the use case description have been annotated and can be found in  `annotated_use_case_diagram_Matchmaking.png`.


Leaderboard and Matchmaking Class Diagram

Pros:

- The diagram effectively separates components into distinct classes such as Leaderboard, MatchmakingLogic and PlayerStats, providing a well organized approach.
- The relationships between classes are mostly clear, indicating how they interact with each other.
- The inclusion of external references such as Profile and AbstractGame shows consideration for system integration.

Cons:

- The matchPlayer method in MatchmakingLogic lacks specific criterias used for matchmaking.
- The rank attribute in Leaderboard and Gamestats lacks clarity and vagueness.
- AbstractGame and MatchmakingLogic dependency lacks context and needs a clear justification to explain why and how they are connected.

Suggestion for improvement and clarification in the use case description have been annotated and can be found in  `annotated_leaderboard_and_matchmaking_class_diagram.png`.

Leaderboard_and_matchmaking_use_case_descriptions

The use cases and the use case diagram are detailed, clear and flow well. The diagram and the description properly maps out the main core functionalities, providing consistency and clarity. Ensures the player and system interaction system components are well covered, including checking leaderboards, rematch requests, viewing profile, statistics etc. Overall great alignment between the description and diagram.



Some use cases could be added to the diagram to maintain consistency and detail so it aligns even better with the use case descriptions. The use case description mentions that players can search for their regional/global ranks on the leaderboards, however the diagram only shows “show leaderboard”. Adding a use case to select the leaderboard options would resolve this issue. The use case also describes how in the matchmaking system, if there is no match found within 45 seconds it will expand the search range, the diagram does not include this “expansion” option. Including these changes could improve clarity and completeness.

Networking

The use case description had great in-depth analysis of each use case. The team extensively covered appropriate scenarios that might occur, such as challenges encountered with stability or wait times.

Some suggestions for improvement would be to consider adding the ability to add certain players into a “Friends List” and request a match with them.

Files:

-  ANNOTATED_use_case_descriptions_networking.pdf
-  ANNOTATED_class_diagram_networking.png

Integration

Class Diagram

Pros:

- Shows an overarching diagram of the system operations
- Explains what each function referenced in the large scale diagram refers to

Cons:

- Diagram may not show full connections between classes
 - Ex. the Authentication functions seem like they would need to be connected to the start and end session functions for Platform so a user can log in once they start a session, but they are not
- Subclasses are shown as “classes” in the diagram which presumably contain their primary functions, or most necessary functions for connections with the other subclasses, however this results in a confusing diagram since it does not show which classes inside each sub function are connecting and, subsequently, which classes rely on functions from others
- Large amount of disparity between the content of the subclasses in the diagram and the content of the separate detailed subclass diagrams
 - The authentication subsystem connection references a function that does not exist, and its closest counterpart is a private function
 - The game logic subsystem contains functions which are not all shown in all the separate game logic diagrams, and some of which do not have an exact counterpart such as endGame()
 - Networking has functions shown that do not have any counterpart in their diagram, and some of the functions that do have a counterpart have different parameter types in the networking class diagram from the integration class diagram
 - GUI section also does not have any matching functions between the two diagrams

Overall, the class diagram is made well. Everything in the diagram seems to connect properly and the functions are even explained for ease of reading. The only issue is that it does not match the other class diagrams created by the other subsections, which means that if implemented it will not operate properly.

Annotated copy included below:

[Annotated Integration Class Diagram](#)

Use Case Descriptions and Diagram

Pros:

- Incredibly well thought out and all cases accounted for
- Links well with the other use case diagrams for their general use cases

Cons:

- Unsure about individuals being able to choose their server
- Need more clarity on what details are kept in match history
- Could expand diagram to include more, but well done for a broad scale given there are other diagrams

Annotated Use Case Descriptions and Diagram below:

[Annotated Integration Use Case Descriptions and Diagram](#)

Connect Four

Class Diagram

Pros:

- Covered all the aspects need for the class diagram
- Implemented useful features in the diagram as well (chatbox)
- Used interfaces to limit dependencies in the code

Cons:

- Repetition in the code between different game diagrams
- Overcomplication of the diagram
 - Using an interface for one function
- Errors in arrow usage and interface implementation
 - Having an interface use a class that only has private variables and no functions (an interface cannot extend a class which would be the only way those variables could be used)
 - No arrow specification beyond association results in a confusing diagram and difficulty identifying the degree of connection between parts
 - Certain sections are selected to be interfaces when the purpose they are required for obligates them to be classes

Link to file for annotations on diagram:

[Annotated Class Diagram Connect4](#)

Use Case Descriptions and Diagram

Pros:

- Well-done and clear, all sections are thoughtfully filled out
- Added system actor to handle majority of the operations after the use case was accounted for
- Diagram used proper syntax

Cons:

- Only use one player as the primary actor for all the functions
- Do not have a system to handle, or account for, the fact that a player may leave the game part way through
- Chat message system may benefit from connecting to a network actor for full operation since it will require a direct connection to another player not through the board

Link to files for annotations on Descriptions and Diagrams:

[Annotated Use Case Descriptions Connect4](#)

[Annotated Use Case Diagram Connect4](#)

ConnectServer_Diagram

1. Clear Representation of Actors

- The **User** and **Server** are clearly represented as actors, making it easy to understand the interaction between them.

2. Logical Flow of Connection Process

- The diagram covers essential steps such as:
 - **Initiate Connection Request**
 - **Validate Connection Requirements**
 - **Monitor Connection Status**
 - **Establish Connection**
 - **Handle Connection Errors**
- These steps ensure a complete and structured flow for the connection process.

3. Error Handling Consideration

- The diagram accounts for **Connection Errors**, which is crucial in a real-world application to manage disconnections or failed connections.

Cons:

1. Complex and Cluttered Layout

- The diagram is somewhat cluttered with overlapping arrows and multiple dependencies. This can make it harder to follow the logical flow, especially for someone new to the system.

2. Redundant or Unclear Relationships

- Some relationships might be redundant or could be simplified. For instance:
 - The direct link from the **User** to **Monitor Connection Status** could be handled through **Establish Connection** instead.

3. Missing Disconnection Handling for User

- The **disconnect** action is shown, but it's unclear whether the user can manually trigger a disconnect or if it only happens due to errors.

Suggested improvements:

1. Improve Diagram Readability

- Rearrange elements to reduce overlapping arrows and make the flow clearer.
- Place **User-related actions** on one side and **Server-related actions** on the other for better organization.

2. Clarify Connection Type

- Add annotations or a legend to specify if the connection is through **WebSockets**, **TCP**, or **HTTP**.

3. **Explicitly Show Reconnection Conditions**

- Define conditions that trigger **auto reconnect** (e.g., network failure, inactivity timeout).


4. **Enhance Disconnection Logic**

- Differentiate between **user-initiated disconnection** vs. **server-side disconnection** for more clarity.
-



Go

The use case diagram for the game Go is consistent with the corresponding use case description, and the diagram covers all aspects of functionality that has been written in the description. Descriptions are clear and concise, and the game logic flow is linear and succinct. This enhances the overall clarity and coherence of the design documentation. The core functionalities of the boardgame Go have been accurately covered. Game logic including initialization and termination of a match have been described. The use case description also acknowledges and considers future improvements, allowing for the application to evolve through long-term usage.

Feedback and suggestions for improvement or clarification in the use case description have been provided by direct annotation on the document itself (see

 ANNOTATED_use_case_description_Go.pdf). In general, the use case description and diagram did not describe a second player. Further suggestions to the game logic of Go include implementation of a second player, whether it is another human player or AI bot. In addition to the current implementation of game termination logic, there should be an “Exit” or “Restart” button introduced for players who want to terminate the game early.

Files:

-  ANNOTATED_use_case_description_Go.pdf
-  ANNOTATED_use_case_diagram_Go.png

Chess (WILL ADD THE SUGGESTIONS FOR IMPROVEMENT)

Class Structure Diagram:

The class structure diagram is overall well designed, easy to read and covers the majority of the aspects needed for a class structure diagram. It properly represented the basic components of chess, tracking where pieces are moved, what players have taken, and the castling rule. The chess class also correctly manages the state of the board.

There were not any clear specifications regarding how the game would handle special movements. The class structure supports how it would account for basic movements but it did not account for the special moves. In addition the implementation arrow is pointing the wrong way, as only a class can implement an interface, the arrow should point from the Piece class to the IPiece interface. The design of the class structure diagram does not check the status of the game whether the player is in check or checkmate. This is crucial as it validates the legality of a player's move.

Use Case Description and Use Case Diagram:

The use case description and the use case diagram is well detailed, structured, and very easy to read. The use case diagram includes all the proper actors involved and the basic functionalities for players to select/deselect spots on the gameboard, make basic moves and switch players based on whose turn it is. The use case descriptions are clear, concise and for the majority aligns well with the use case diagram. The main functionalities of the game chess have been covered. The diagram follows the sequence of a player making their move ensuring the selected piece is moved to a valid spot on the game board. In addition the "switch Active Player" allows for a smooth transition for the next player to take their turn.

Although the use case descriptions were well done, there could be some improvements. For instance there were no use cases for options to play the game in single player or multiplayer mode. This is important as if a second active player does not exist the first active player can choose to play alone against an AI bot if implemented. Another use case could have been added to describe how the AI bot will take control and become the active player for the game. In the diagram it does not portray how special moves should be handled. The use case "Make Move" describes how all moves follow the basic rules, however they do not mention anything about the special moves. In addition the use case descriptions mention players should be able to see the valid move options. This should be included in the use case diagram so it matches. The diagram should also include a section where the game is able to verify if a move is valid or not. This will help with the game logic's flow and will enhance the clarity of the overall design.

Use_case_diagram_chat

The diagram has a clear representation of arrows and the proper use of <<includes>> and <<extends>>. The diagram also includes the core functionalities including sending, receiving, deleting, editing messages, viewing chat history etc.

The diagram looks very complex as there are many overlapping arrows making the diagram difficult to read. Improving the organization and grouping related functionalities closer together will reduce the messiness providing clarity. It would also be beneficial to provide how the server would handle storing messages. There is also no section mentioning how the system would handle failed messages. It does not send a specific message to notify the reasoning behind the failed message delivery. Finally it would be beneficial to include a blocking or reporting mechanism to protect users. Adding a blocking and report use case could be considered as improvements.

use_case_diagram_UserData

The diagram covers the basic authentication steps, ensuring proper user validation. Good usage of the <<includes>> and <<extends>> arrow relationships. Included the basic actors for the functionalities needed including the user, server and database.


Overlapped arrows could be avoided by rearranging the use cases to reduce messiness and overlaps. Some extra system commands could be implemented such as password recovery options, and multifactor authentication for extra security and privacy. Use cases could be added to the diagram to portray these new added features. It would also be beneficial if the diagram included a section for handling locking accounts after many failed login attempts. An extended use case could be helpful instead of it only throwing an error over and over again. The diagram could also include a log out use case if the user prompts to switch accounts. This can be implemented into the diagram by including a log out use case.


Tic Tac Toe

Tic Tac Toe Use Case Description

The use case description did not fully address the possibility of a tie, and the flow of game termination and subsequent actions could be clarified.

Additionally, the handling of player disconnections should be more clearly defined, including whether the remaining player faces any penalty and how a new match will be chosen or if an AI will be substituted.

Suggestions for improvement and clarification in the use case description have been highlighted directly within the document  annotated_use_case_description_tictactoe.pdf .

For the associated diagram, there is a “Board” interface and “Board” class that implements this interface. A suggestion is to change the interface class to “BoardInterface”, so that these classes have distinct names. See here:  ANNOTATED_class_structure_diagram_tictactoe.png

Query


Query Use Case Diagram

Pros:

- Key use cases are defined.
- <<include>>, <<extend>> relationships demonstrate flexibility in the system design.
- Actors are well-defined and connected to the use cases.

Cons:

- Even though handling connection issue is included, additional error handling measures would improve the user experience.
- The roles of the server and database could be explained in more detail to improve clarity and understanding.
- It would be helpful to include handling for invalid query submissions since the current design only considers “Validate Query” and “Redirect Query” without addressing potential failures or errors. The way it is managed is unclear and could be better defined.

Suggestion for improvement and clarification in the use case diagram have been annotated and can be found in  annotated_use_case_diagram_Query.png

Synchronization


Synchronization Use Case Diagram

Pros:

- The diagram provides a great overview of the synchronization process, highlighting key interactions between actors and use cases.
- Network latency and resolve sync conflicts handles real issues and makes the system more reliable.

Cons:

- Some of the relationships between use cases lack clarity about their conditions or triggers.
- Synchronization steps could be more detailed, missing key processes.

Suggestion for improvement and clarification in the use case diagram have been annotated and can be found in  annotated_use_case_diagram_Synchronization.png .