

Draw it or lose it

# **CS 230 Project Software Design Template**

Version 3.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc115077317)

[**Table of Contents 2**](#_Toc115077318)

[**Document Revision History 2**](#_Toc115077319)

[**Executive Summary 3**](#_Toc115077320)

[**Requirements 3**](#_Toc115077321)

[**Design Constraints 3**](#_Toc115077322)

[**System Architecture View 3**](#_Toc115077323)

[**Domain Model 3**](#_Toc115077324)

[**Evaluation 4**](#_Toc115077325)

[**Recommendations 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 3.0 | 04/21/2025 | Kyndal Erickson | Final Draft |

## [Executive Summary](#_sbfa50wo7nsh)

The purpose for this software design document is to outline the proposed solution for the development of the web-based game application, “Draw it or Lose it”, for client The Gaming Room. The solution requires the rendering of images from a library of stock drawing as clues, allowing multiple teams to participate in four rounds of gameplay. By expanding this game from an Android based to a web-based platform, it can then be accessible across various devices and provide enhanced game experience. The proposed design takes into consideration the client’s needs, aiming to deliver a user-friendly and engaging gaming application.

## Requirements

* The game should be accessible through a web-based platform, ensuring compatibility with different devices and operating systems
* Each game should support the existence of one or more teams with the ability to assign multiple players to each team.
* A singleton creation pattern must be utilized to support only one single instance of the game in memory using unique game, team and player names, and should allow players to check for name availability during game creation.
* Each game round must have specific time limits, i.e. one minute per round, with drawings being revealed progressively until completion at time limit.
* If the initial teams fail to guess the drawing within the given time limit, then the remaining teams get the chance to guess within a 15-second time limit to solve the puzzle.

## [Design Constraints](#_2et92p0)

* *Web-based Distributed Environment: The Gaming Room application must be designed for the operation on a web-based platform, introducing constraints related to network communication, security, and compatibility with various web browsers/devices.*
* *Uniqueness: The service should enforce uniqueness of game, team and player names/IDs to prevent naming conflicts.*
* *Single Instance limitation: To ensure the game operates as intended, the design should account for the limitation of only one instance of the game service being active in memory at any given time.*

## [System Architecture View](#_ilbxbyevv6b6)

Please note: There is nothing required here for these projects, but this section serves as a reminder that describing the system and subsystem architecture present in the application, including physical components or tiers, may be required for other projects. A logical topology of the communication and storage aspects is also necessary to understand the overall architecture and should be provided.

## [Domain Model](#_8h2ehzxfam4o)

The following diagram represents a parent-child relationship between the base class “Entity” and its children classes: Game, Team, and Player. The diagram also includes a composition relationship between each child class. The Entity class contains common attributes: id’ and ‘name’ and common behaviors that can be shared amongst all subclasses through extension. Each game, team, and player will have unique ids and names, representing the main entities within the game application (a game consisting of multiple teams and a team consisting of multiple players).

Through the relationship between the GameService and Game class, the GameService class manages the lifecycle of Game instances and holds references to multiple Game objects. The Game class also holds a composition relationship with the Team class, as well as the Team class holding a composition relationship with the Player class.

The main function resides in the ProgramDriver class and serves as an entry point to the application. The GameService singleton instance is created, indicating that there will only be one instance of the GameService class through the application. The program driver class is responsible for adding games, teams, and players using the GameService instance. Lastly, the ProgramDriver class has a dependency on the SingletonTester class, as used shown with the <<uses>> arrow.

The UML class diagram demonstrates several OOP principles, such as inheritance, encapsulation, and abstraction. The diagram demonstrates inheritance between the Entity Parent class and its multiple children classes (Game, Team, and Player), enabling the children classes to inherit attributes and behaviors from its parent class. The diagram also demonstrates encapsulation, with the GameService class only providing methods to interact with its list of existing games and constructor. This approach helps to maintain data privacy by ensuring only one instance of the game gets created. Lastly, abstraction is represented in the diagram through the way the classes manage games, teams and players, concealing unnecessary implementation details, focusing only on the critical interactions.

**"The Gaming Room UML diagram. The top of the diagram is labeled as com dot gamingroom. Test boxes are placed in two layers. The first layer has three text boxes and the second layer has four of them. In the first layer, the 'ProgramDriver' textbox points to 'SingletonTester' textbox. The 'ProgramDriver' textbox contains the text 'asterisk main round brackets.' The 'SingletonTester' textbox contains the text 'asterisk testSingleton round brackets.' The arrow between these two text boxes are labeled 'open two angle brackets uses close two angle brackets'. In the second layer, there are 'GameService', 'Game', 'Team', and 'Player' text boxes. The 'GameService' textbox has texts arranged in two layers. The first layer contains games colon List open angle bracket Game close angle bracket, nextGamesId colon long, nextPlayer Id colon long, nextTeamId colon long, and service colon GameService. The second layer contains GameService round brackets, getinstance round brackets colon GameService, addGame open parenthesis name colon String close parenthesis colon Game, getGame open parenthesis id colon long close open parenthesis colon Game, getGame open open parenthesis name colon String close open parenthesis colon Game, getGameCount round brackets colon int, getNextPlayerID round brackets colon long, and getNextTeamId round brackets colon long. The 'GameService' box is connected with the 'Game' textbox with a line labeled 'zero dot dt dot asterisk'.  The 'Game' textbox also contains text in two layers. The first layers contains the text teams colon List open angle bracket Team close angle bracket. The second layer has Game open round bracket id colon long comma name colon String close parenthesis, addTeam open parenthesis name colon String close parenthesis Team, toString round brackets colon String. The 'Game' textbox is connected with the 'Team' textbox with a line labeled 'zero dot dt dot asterisk'. The 'Team' textbox also contains text in two layers. The first layers contains the text players colon List open angle bracket Player close angle bracket. The second layer has Team open parenthesis id colon long comma name colon String close parenthesis, addPlayer open parenthesis name colon String close parenthesis colon Player, and toString round brackets colon String. The 'Team' textbox is connected with the 'Player' textbox with a line labeled 'zero dot dt dot asterisk'. It contains the text Player open parenthesis id colon long comma name colon String close parenthesis and toString round brackets colon String. The 'Game', the 'Team, and the 'Player' boxes point to the 'Entity' textbox in first layer. The 'Entity' textbox contains text in two layers. The first layer has the text id colon long and name colon String. The second layer has Entity round brackets, Entity open parenthesis id colon long comma name colon String close parenthesis, getId round brackets colon long, getName round brackets colon String, toString round brackets colon String.**

## [Evaluation](#_2o15spng8stw)

Using your experience to evaluate the characteristics, advantages, and weaknesses of each operating platform (Linux, Mac, and Windows) as well as mobile devices, consider the requirements outlined below and articulate your findings for each. As you complete the table, keep in mind your client’s requirements and look at the situation holistically, as it all has to work together.

In each cell, remove the bracketed prompt and write your own paragraph response covering the indicated information.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Pros: Stable and secure foundation for server hosting; robust ecosystem; developer friendly.  Cons: Limited scalability; less graphics memory; closed-source operating system. | Pros: Open-source operating system; flexible; has a wide range of software tools; highly scalable; known for its stability and security. | Pros: Software compatibility; strong developer ecosystem; robust support and documentation; regular updates; high software stability.  Cons: More known security vulnerability; closed-source operating system. | Pros: Portability; touchscreen and gesture-based interaction.  Cons: Limited screen size; varied hardware capabilities. |
| **Client Side** | Pros: User-friendly interface reduces learning curve; Cons: Increased development costs, time, and may require diverse expertise; high cost for hardware; small support community. | Pros: Free licensing; mostly free tool licensing; customizable.  Cons: Diverse expertise needed for different clients; steeper learning curve. Some GUI limitations and hardware compatibility. | Pros: Broad hardware support; entry-level friendly; user-friendly interface; customizable; multitasking ability.  Cons: High licensing costs; minimal control of privacy settings. | Pros: High accessibility; affordable app and game downloads.  Cons: Pricey in-game or app purchases; less processing power and battery capacity. |
| **Development Tools** | IDEs/ Code Editors:  Xcode, Visual Studio Code, Docker Desktop for Mac; Mobile/ Game Development: Flutter; React Native, Unity, Godot; Design: Figma, GIMP. | IDEs: VSCode; CLion; GNOME Builder; Vim. Web Development: Docker; Apache; WSL (dual booting).; GUI Frameworks: Qt; Electron; Flutter. Design: GIMP; Inkscape; Krita. | IDEs: Visual Studio, Docker Desktop for Windows, IIS. | Game Engines: Unity, Unreal Engine, Godot;  IDEs/ Code Editors: Visual Studio, Visual Studio Code, Android Studio; Deployment tools: Android SDK and NDK; Xcode. |

## Recommendations

1. **Operating Platform**: To allow the Gaming Room to expand Draw It or Lose it to other computing environments, adopting a web-based platform is recommended. Doing so would allow the game to be utilized and played on various devices with web browsers such as desktops, laptops, tablets, and smartphones. By leveraging web technologies, the game can reach a broader audience and provide a consistent experience across different platforms.
2. **Operating Systems Architectures**: For the chosen web-based operating platform, the architecture will primarily revolve around client-server and web-based technologies. The client-side architecture will involve the use of HTML, CSS, and JavaScript to render the game interface and handle user interactions. On the server side, multi-tier architecture can be employed consisting of presentation, application, and data tiers.
3. **Storage Management**: An appropriate storage management system would be a combination of a relational database management system (RDBMS) and cloud storage devices. RDBMS can handle structured data related to game progress, user profiles, and game statistics. Cloud storage services can be used for storing media assets, such as stock images, ensuring scalability and accessibility across platforms.
4. **Memory Management**: The recommended web-based operating platform utilizes automatic memory management techniques provided by modern web browsers. Browsers employ garbage collection mechanisms to handle memory allocation and deallocation, free developers from manual memory management. This helps ensure efficient memory utilization and prevents memory leaks.
5. **Distributed Systems and Networks**: To enable communication between various platforms, the game can utilize distributed software architecture and leverage network connectivity. This can be achieved by implementing a centralized server or cloud infrastructure that serves as a communication hub between game clients. The server can handle game synchronization, real time updates, and message exchange between players across different devices. The system should account for network connectivity issues, such as intermittent outages or low bandwidth, by having appropriate error handling and synchronization.
6. **Security**: To protect user information on and between various platforms, several security measures can be implemented. The recommended web-based operating platforms should support secure communications protocols. User authentication and authorization mechanisms, such as username and password, can be implemented to control access to game features and user profiles. Additionally, data encryption techniques can be used to safeguard sensitive user information stored in databases or transmitted across the network.