

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## 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 |
| --- | --- | --- | --- |
| 1.0 | 03/21/25 | Chris O’Dell | Executive Summary and Domain Model initial upload. |
| 1.1 | 04/04/25 | Chris O’Dell | Evaluation initial upload. |

**Instructions**

Fill in all bracketed information on page one (the cover page), in the Document Revision History table, and below each header. Under each header, remove the bracketed prompt and write your own paragraph response covering the indicated information.

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room would like to recreate their current Android exclusive game, Draw It or Lose It, in a web-based release so that their game can server multiple platforms. The game is comprised of multiple teams that attempt to guess what image is being drawn. The application will render images from a library of drawings and players on the current team’s turn will attempt to guess what is being drawn. Each game will consist of four rounds with each round lasting one minute each. Drawings will be drawn at a steady rate and drawings will be fully rendered after 30 seconds. If the current team does not provide the correct answer within the one-minute mark, the remaining teams each have 15 seconds to provide one guess to solve the puzzle.

## Requirements

* Game must be web-based and run on multiple platforms
* A game must have one or more teams involved
* Each team will have multiple players
* Game and team names must be unique
* Only one game instance can exist in memory for each instance of a game, team, or player

## [Design Constraints](#_2et92p0)

* Cross-Platform Compatibility
  + The game must run on web browsers across multiple devices (PC, Mac, mobile, tablets).
  + Should support modern web standards (HTML5, CSS, JavaScript, WebSockets).
* Single Instance Enforcement
  + Only one game instance can exist in memory per game session.
  + Each team and player must have a unique instance in memory.
* Uniqueness Constraints
  + Game names, team names, and player names must be unique.
  + A mechanism must be in place to prevent duplicate entries.
* Real-Time Gameplay
  + Must support real-time multiplayer interactions.
  + Game logic must allow synchronized drawing and guessing.
  + Client-Server design to support multiplayer synchronization.
* Time Constraints & Game Flow
  + Each game consists of four rounds, each lasting 60 seconds.
  + Drawings must be gradually rendered over 30 seconds.
  + The current team has 60 seconds to guess, and other teams get 15 seconds each if the first team fails.
* Scalability & Performance
  + The server must handle multiple active game sessions efficiently.
  + Minimize latency to ensure smooth real-time gameplay.
* Security & Fair Play
  + Prevent duplicate connections for a single player.
  + Restrict access to game logic to prevent cheating=======.
* Networking & Communication
  + WebSockets or other real-time protocol must be used for instant updates.
  + The server must manage game states, player actions, and drawing progress.
* Data Storage & Retrieval
  + Maintain a database for game history, player data, and team records.
  + Efficient querying to enforce uniqueness constraints.

## [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 Gaming Room UML diagram provides a visual representation of the relationships between each class in the Draw It or Lose It game application. ProgramDriver provides the application’s main function and uses the SingletonTester to test the singleton instance of the GameService class. GameService manages the Game, Team, and Player classes and operates as a singleton so that only one GameService instance can ever be instantiated within the application. GameService maintains a list of Games and manages the unique IDs of games, players, and teams and allows global access to getters of the various variables within the application as well as adding new games. The Game, Team, and Player classes are all subclass, or child classes, of the Entity super, or parent, class. The Game class contains a private list of Team objects, a public constructor, a public method to add teams to the list using an iterator, and a public toString function to display the game id and name. The Team class contains a private list of Player objects, a public constructor, a public method to add players to the list using an iterator, and a public toString function to display the team id and name. The Player class contains a public constructor and a public toString function to display the player id and name. The Entity class contains a private id, name, and constructor as well as a public constructor, two public getters for id and name, and a public toString function to display the entity’s id and name (these are overridden by the subclasses).

The OOP principles demonstrated within the UML diagram include encapsulation through the use of private attributes and public getter methods to restrict and provide controlled access to objects’ data. Inheritance is used with the Entity class being a base class for Game, Team, and Player classes, providing code reusability through common variables such as id, name, and getter methods for the variables. Polymorphism is used with function overrides with the toString methods of each class as well as addTeam and addPlayer methods allowing dynamic object creation.

**"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** | Unix-based OS, great security, robust dev ecosystem. Higher hardware cost, and less scalability. Not ideal for server-side hosting as there are not many options and proprietary ecosystem is limiting. | Cost-effective, flexible, and stable open-source OS. Strong security, scalability, and vast community support. The command-line interface can have a steeper learning curve. | User-friendly GUI and strong integration with the Microsoft ecosystem, great for .NET applications. Broad hardware compatibility and good support, it comes with higher licensing costs and has historically faced more security concerns. | Mobile devices have limited processing power and network scalability. Lacks necessary features and resources for reliable server-side web app hosting. Other OS options should be considered. |
| **Client Side** | High cost for Mac proprietary hardware and licenses. Development time can be higher due to strict Apple standards. Requires knowledge of macOS specific frameworks such as XCode/Swift. | Low development cost as Linux is open source. Development time can be high, depending on the amount of customization and compatibility needed. High level of expertise required working in Linux environments such as shell-scripting and kernel-level configuration. | Moderate cost for licensing and hardware. Less development time needed than other options. Requires high-level of expertise in .NET framework, Windows API, and C#/C++ languages. | Costs vary depending on the mobile ecosystem (iOS/Android). Platform-specific development can increase cost (licensing/hardware) and time if using separate SDKs. Expertise in mobile development is required as well as mobile-specific UI/UX design and design requirements/limitations for varying devices. |
| **Development Tools** | Mac development mainly use XCode (Mac native), AppCode, VSCode for development tools/IDES. Swift is Apple’s primary coding language for iOS devices. Objective-C, Python, and JavaScript are also relevant languages. | Relevant languages for Linux include C/C++, Python, Java, Ruby. Relevant IDEs include Eclipse, VSCode, Vim. | Windows has a plethora of relevant languages including C#, C++, Python, Java, JavaScript. Relevant IDEs for use on Windows include Visual Studio, VSCode, Eclipse, and the JetBrains suite of IDEs and tools. | The main languages used for mobile development include Swift (iOS), Kotlin and Java (Android). When developing with Swift, XCode is an excellent IDE, and for Android, Android Studio and VSCode are excellent options. |

## Recommendations

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

1. **Operating Platform**: The Linux operating platform is recommended due to its relatively lower cost, flexibility, scalability, cross-platform friendliness, and vast documentation and support. Specifically, Linux servers are an excellent choice for flexibility and scalability which are two important factors for The Gaming Room’s objectives and goals with Draw It or Lose It.
2. **Operating Systems Architectures**: Linux supports several architectures, including a modular kernel architecture, which is ideal for The Gaming Room. This architecture will allow dynamic loading of modules for specific hardware or features which allow the system to scale with ease. It also supports multiuser and multitasking, supporting concurrent players and server tasks effectively.
3. **Storage Management**: The Logical Volume Manager is recommended for Linux due to its flexible disk management, enabling partition resizing and efficient space allocation which is crucial for handling dynamic game data. An efficient and reliable backup system is necessary for nearly any online video game and utilizing tools like AWS for cloud-based backups or other tools like rsync will provide The Gaming Room with trusted data recovery.
4. **Memory Management**: Linux uses various memory management techniques including virtual memory which allows applications to use more memory than physically available through swap space as well as memory allocation to manage memory caches which improves performances under heavy loads.
5. **Distributed Systems and Networks**: To allow Draw It or Lose It to communicate across different platforms, a distributed architecture using APIs and network services is necessary. RESTful APIs enable real-time communication between clients and servers. WebSockets are also useful for games that require constant updates. Cloud integration using AWS or Azure allows for scalable distribution. Backend services and API endpoints hosted on Linux servers can serve requests from various platforms.
6. **Security**: Linux offers several security features to protect user data such as user authentication and access control through Linux’s permission model and user/group controls to restrict access to certain users. Encryption though SSL/TLS for all client-server communication enhances security of sensitive data through hashing. Firewall configurations help to add an extra layer of security and threat detection.