

Draw It or Lose It

# **CS 230 Project Three**

# **Software Design Template**

Version 2.0

## 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 | 10/12/2025 | Jonathan Dwyer | Updated executive summary, design constraints, evaluations, and recommendations |
| 2.0 | 10/26/2025 | Jonathan Dwyer | Updated evaluations and recommendations. |

**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)

This report presents the outcome of our examination and the intended design of Draw It or Lose It, a multiplayer web game built for our client, The Gaming Room. We assessed the performance of the different platforms that the game can run on, such as desktop platforms (Linux, Mac, and Windows) and mobile platforms (Android and iOS). Our findings demonstrate that the solution performs smoothly as well as reliably on any device. The intended design centers on making the solution highly accessible, scalable, as well as giving the player an enjoyable experience. Utilizing current web technologies enables us to guarantee that Draw It or Lose It is easy to use, easily accessible, as well as enjoyable to use anywhere in the world. We also strive to achieve the client’s objectives of having a great time as well as playing across platforms.

## Requirements

1. The game will have the ability to have one or more teams involved.
2. Each team needs to be able to have multiple players assigned to it.
3. The game will allow teams to create unique names with checks to determine if the name is currently taken.
4. Only one instance of each game, distinguished by a unique identifier, can exist at a time.
5. 4-round game model with 1-minute rounds, at 30 seconds left, full image revealed.
6. If the team fails to correctly guess the image, the next team will have 15 seconds to guess.

## [Design Constraints](#_2et92p0)

1. Network and Bandwidth issues
   1. As a web application, a stable network will be key for smooth gameplay. High-quality graphics and multiple players can increase bandwidth needs, causing lag or delays for users with slower connections. To fix this, the game should compress data, optimize media files, and adjust graphic quality based on network speed.
2. Unique Name Management
   1. There will need to be a repository to store team names to enforce unique names. This data will also need to be protected to prevent unauthorized changes and maintain integrity.
3. Single Game Instance
   1. Running only a single instance of the game at a time to keep the processing requirements low. It also avoids synchronization problems and keeps performance steady. While this limits scale early on, it creates a strong base for future upgrades like multi-instance or cloud scaling.

## [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 UML class diagram for the Draw It or Lose It application provides a representation of the arrangement of the system's components, as well as their connections. The superclass, which is Entity, facilitated a means (attributes) to place common id and name attributes in Entity that Game, Team, and Player inherit from to minimize duplicate information and provide attribute conformity.

The relationships present in the diagram utilize both inheritance (is-a) and composition (has-a) principles. The GameService class will manage multiple instances of the Game object, the Game object will contain multiple instances of the Team object, and the Team object will manage multiple instances of the Player object. This presents a clear hierarchy; GameService → Games → Teams → Players. The ProgramDriver class is considered the instantiator class, which instantiates a singleton instance of GameService and has a dependency upon the SingletonTester class.

The class diagram demonstrates core principles of object-oriented design. The inheritance relationship allows subclasses to share structure and behavior from the Entity class. Encapsulation allows for classes like GameService to keep their state private while exposing necessary methods. Abstraction allows the system to manage the game, teams, and players without having to include unnecessary detail. These object-oriented principles create better designs that are efficient, maintainable, and address all software requirements.

**"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** | Provides a stable UNIX-based platform for server-based deployment that imposes fewer technical constraints. User-friendly GUI. The hardware and licensing costs are on the higher end of options. Software has many open source options on Linux. | Provides a cost-friendly open source OS. Numerous open source utilities and libraries. Simple and customizable modular configuration. Great for scalability. The navigation can be difficult. The GUI has limitations compared to other OS options. Most utilities are operated via the command line. Potential to have high cost as product size scales. Access control is easily circumvented. | Provides an easy-to-use GUI environment. An expansive ecosystem providing a wide variety of capabilities and support. Lower cost to no cost service fees. Supports a wide variety of very large Java APIs. Higher potential of security issues based on known threats. Microsoft's web framework increases the technical constraints. Layered architecture is very bloated and can lead to performance issues when scaling. | Several frameworks available that promote scalability and performance for web hosting. Provides portability. The majority of the deployment tools are multi-platform supported. Varied hardware capabilities based on the device. Typically requires a connection to a computer with internet to support DNS functions. |
| **Client Side** | Provides a user-friendly and simple environment. The hardware and licensing costs are on the higher end of options. | Provides a cost-friendly open source OS. Open source alternatives to common Windows/Mac tools. Learning curve on OS requires learning to gain basic ability/expertise to navigate. | Widely known OS providing a familiar user experience. Significant license costs associated with tools. | Provides flexibility with updates and feedback. Responsive design. Have the ability to utilize built-in features. Implementation can be difficult across mobile OS’. Resource consumption management. |
| **Development Tools** | There is a wide variety of tools and languages supported for full-stack development. Native robust development environment using ZSH/BASH. Common languages include C, C++, Java, and JavaScript. | A wide variety of tools and languages are supported for the full-stack. Robust development environment. Common languages include C, C++, Java, and JavaScript. Development environment setup could be difficult based on the level of Linux expertise. | A Wide variety of tools and languages are supported for the full-stack. Robust development environment using C# and .NET. Other common languages include C, C++, Java, and JavaScript. | IDE-based emulators are available to test on multiple platforms. Uses robust dynamic languages like Kotlin/Swift/Java. Depending on the mobile OS, the supporting tools can get expensive. |

## 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**: For The Gaming Room to run efficiently both in resource allocation and cost, I would recommend using the Linux OS. The key benefits this OS would provide would be reduced operating costs due to the amount of open-source utilities and tools, and the inherent flexibility and security provided.
2. **Operating Systems Architectures**: In the modern three-tier architecture, the middle layer (or application layer) is responsible for executing business logic while also acting as a mediator between the presentation and database layers (TechTarget, 2024). Building on this model, serverless computing decomposes the application layer into modular, event-driven components that will allow them to scale independently. Using components like authentication services, content distribution systems, and business logic functions will allow The Gaming Room to only use what is needed and only require payments for those services (AWS, 2023). Similar to this modular design, Linux’s kernel serves as the OS’s core, managing essential functions and tasks scheduling processes, and managing memory. You can then add optional modules dynamically to increase functionality without having to recompile the entire system (Love, 2015). This minimizes the storage requirement while simultaneously enhancing performance by removing reliance on the slower inter-processes found in OS designs. Finally, due to Linux’s reliance on user space system libraries to perform operations without granting direct kernel access, you can keep your environment secure during execution while preventing unauthorized hardware interaction (Red Hat, 2019).
3. **Storage Management**: The recommended storage management for this would be a cloud-based solution that would allow The Gaming Room to only use the space needed for the game where it currently is, but would allow it to be expanded as more users start playing this game.
4. **Memory Management**: Utilizing Java as our primary language, it has a very robust native memory management suite, including garbage collection. This feature clears memory if objects currently stored are no longer being referenced.
5. **Distributed Systems and Networks**: Due to the need to provide this game cross-platform, it will be hosted on a web application using a browser to access. Preferably, it would focus on compatibility with Google Chrome as it is available on all OS’s.
6. **Security**: The primary security method will be username and password with RBAC to limit access to necessary areas. The basic roles would be admin, user, and guest, each with its own level of access and permissions. This will ensure that admins will be able to manage the users and guests as needed, while the users and guests will not be authorized to manage at that level. Data encryption will also be used to ensure sensitive data is stored without the threat of that data being retrieved by unauthorized personnel.

**References**

*Gillis, Alexander S. “What Is a 3-Tier Application Architecture?” Search Software Quality, TechTarget, 22 Oct. 2024, www.techtarget.com/searchsoftwarequality/definition/3-tier-application#:~:text=A%20three%2Dtier%20application%20architecture,tier%20and%20a%20data%20tier.*

“Serverless Computing.” *Amazon Web Services*, n.d., aws.amazon.com/serverless/.

Love, Robert. *Linux Kernel Development*. Addison-Wesley, 2015.

“What Is the Linux Kernel?” *Red Hat - We Make Open Source Technologies for the Enterprise*, *27 Feb. 2019,* www.redhat.com/en/topics/linux/what-is-the-linux-kernel