

The Gaming Room

# **CS 230 Project Software Design Template**

Version 1.2

## 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 | 05/24/2025 | Dennis Nierzwicki | Initial software design briefing |
| 1.1 | 06/08/2025 | Dennis Nierzwicki | Updated evaluation |
| 1.2 | 06/22/2025 | Dennis Nierzwicki | Addendum to 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)

Our firm is working with The Gaming Room to develop a web-based game, Draw It, or Lose It, to expand and serve multiple platforms where currently only Android users can use the application. The application renders images from a stock library as clues and players on teams must decode the puzzle, phrase, or thing.

## Requirements

* Support for one or many teams
  + Must contain unique team names
* Support for one or many players
  + Must contain unique names
* Only one instance of the game can exist in memory at any given time

## [Design Constraints](#_2et92p0)

* Web-based system will need to account for a variety of browsers, screens, and devices.
* Browser architecture can change through the course of development.
* Accessibility is an important factor to consider when designing the UI/UX in regard to internet speeds, devices and browser support.
* Hosting considerations. Shared, cloud-based, or dedicated structures depending on budget.
* Database considerations. Where are the images stored? Are we compressing on load or will the current database be compressed and serve optimized WebP? Or do we allow more browser support and serve PNG/JPG but with larger file sizes?
* What frontend architecture are we using?

## [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)

ProgramDriver is the main entry point of the application and uses the utility class, SingletonTester. SingletonTester can be removed or reused because it is just verifying the singleton is working as it is supposed to.

GameService has a zero-to-many relationship to Game, because it can manage many games, however, GameService is a singleton and can only have one instance of itself. Game has a zero-to-many relationship to Team, which also has a zero-to-many relationship to Player.

Entity ties it together with encapsulation because it holds the private ID and NAME fields with public getters to access them. This means Game, Team, and Player don’t expose their inner workings but rely on inherited behavior from Entity.

Game, Team, and Player all extend the Entity class and use its setter and getters. This is a perfect example of inheritance as seen in the UML diagram below with the arrow pointed up at Entity from each one.

With Entity providing the ID and Name, this means there cannot be any duplicates. i.e. if we stored the ID and NAME in Game, Team, or Player class. It abstracts those details from the three classes extended onto it.

Polymorphism is more difficult to point to in this project. Each class uses method overriding particularly with toString() while relying on shared behavior with super. Each class is represented as its own but inherit from the same Entity base class. So, they behave differently while still sharing a common interface.

**"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** | While MacOS can host web-applications, it has steep licensing and hardware considerations. OSX Server was built for server deployment but was discontinued. This would probably be the most expensive solution and not offer much scalability (Most Macs only offer minimal upgradability such as RAM). Mac Pro may offer more upgradability, but starts at $7,000 | Linux is not going to have licensing issues since it’s open source. It can also be configured to offer the best performance and scalability. Hardware can vary and be scaled based on current needs. Linux also has several important security features. This could be the cheapest option depending on initial hardware. | Windows has some of the same disadvantages as a Mac, like licensing. It is also resource-heavy out-of-the-box. Not to say there aren’t Microsoft enterprise environments. Because Microsoft is less flexible than Linux in performance, Hardware may be more expensive to get similar performance. | Mobile devices aren’t a consideration. Low processing power, storage, network reliability makes mobile devices the worse of the four. Further research suggests, it IS actually possible to use a mobile device as a server, but not feasible at all for this situation. |
| **Client Side** | We are developing web-based and MacOS supports a variety of browsers. We will still need to run Safari testing as it is the native browser. | Web-based application will be fine on a variety of browsers that are supported in Linux. However, this will most likely be the least used system. | Second to mobile devices, Windows provides built-in browser and supports all the major browsers. PCs are normally a cheaper option for most of the population, which means more popular. Edge is the default browser and will need testing. | This will be the most used device. The web-application will need to be developed for mobile, dark mode, and touch screens. It will be critical that responsiveness is tested. |
| **Development Tools** | MacOS would be a good fit for development. It works with all the major IDEs like VS Code, IntelliJ, etc. and is Unix-based like Linux. Bash works in terminal without additional adjustments like Windows. MacOS comes preinstalled with the Apple hardware. This, again, could end up being the most expensive option. | This would probably be the best environment. It also works with all major IDEs, Git, Bash and is open source. It can perform well even on a less powerful machine. This could be the least expensive option. | Windows also runs all the most popular IDEs, but it takes a lot more to get Bash and Git to work but can with Windows WSL. It is also resource-heavy, but for development, would be fine. Some developers run Ubuntu (Linux) in a virtual machine on Windows (free). For development, this could be as cheap as the Linux hardware. The actual process of coding is not resource-heavy. | A mobile phone would not be a great place to develop. However, it could be used as a tool to test mobile responsiveness. |

## 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**: If cloud-based hosting is out of the question, the preferred operating platform for this project would be Linux. This open-source OS (operating system) comes with the least cost association, Security and memory management equal to or surpassing both Windows and MacOS without additional licensing costs. Linux can be customized specifically for this project and is exceptionally scalable. Hardware costs would also prove to be more minimal compared to the other OS platforms.   
     
   I would be amiss to not make a cloud-based recommendation though. No or low maintenance, costs that scale based on application performance, secure, reliable, and no hardware constraints. You pay for what you need.
2. **Operating Systems Architectures**: Linux uses the Linux kernel and is UNIX-based. It is a great choice for hosting web applications because it is fast, efficient, and modular. It can be tailored for a specific purpose and allows extensive customization because it is open source (GeeksforGeeks, n.d.). It includes everything we would need to setup and deploy our application.
3. **Storage Management**: Linux comes with powerful built-in storage capabilities. It uses partitions and supports a variety of file systems such as ext4, XFS, and Btrfs. Storage can be expanded with cloud-based solutions or configured with RAID systems (Cycle.io, n.d.). RAID configurations allow redundancy to prevent data loss and/or speed.
4. **Memory Management**: Linux offers strong Java memory management support through the Java Virtual Machine (JVM). The JVM automatically handles garbage collection by removing unused objects, helping developers optimize performance and simplify memory handling (GeeksforGeeks, n.d.). Linux overall system is optimized to provide memory performance with less hardware needs. Again, reducing costs while still providing ample performance.
5. **Distributed Systems and Networks**: Using a Linux server, it would communicate across platforms and devices using HTTPS. The Draw It or Lose It app can use APIs to communicate user actions and game state. Technically, all three systems would perform equally as well in this category.
6. **Security**: Linux includes strong security features out-of-the-box. It supports user permissions, firewalls, and additional modules like SELinux and AppArmor to enhance platform security. HTTPS can be used to secure communication between clients and the server (LinuxSecurity, 2021). This is another category that all three systems would equally perform well.

**REFERENCES**

Cycle.io. (n.d.). Linux storage management. https://cycle.io/learn/linux-storage-management

GeeksforGeeks. (n.d.). Architecture of Linux operating system.

https://www.geeksforgeeks.org/architecture-of-linux-operating-system/

GeeksforGeeks. (n.d.). Java memory management.

https://www.geeksforgeeks.org/java-memory-management/

LinuxSecurity. (2021, May 26). How secure is Linux?

https://linuxsecurity.com/features/how-secure-is-linux