

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 | 06/12/23 | Alexander Creznic | This document expands upon the previous structure for the executive summary, software system requirements, design constraints, and more. The Evaluation section is also expanded to further explain the server side costs, client-side costs, and development tooling required to facilitate a multi-platform, multi-operating system deployment. |

**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 plans to develop a web-based game called 'Draw It or Lose It' with cross-platform compatibility. Initially available only on Android, this game aims to involve multiple teams, each consisting of several members, in a sequence of four one-minute rounds. During each round, a picture is randomly chosen from a collection of images, and one team endeavors to guess the image within the allocated time. If the answer is not provided, every member of the opposing team has a 15-second window to respond.

## Requirements

1. *The gameplay can involve multiple teams.*
2. *The gameplay can involve multiple teams.*
3. *Unique names are required for both the game and teams. Users should have access to a list of names to verify availability for their desired usage.*
4. *Only one instance of gameplay, or save state, can be stored in memory simultaneously. Additionally, each player, team, and name instance must be distinct to meet the specifications of requirement 3.*

## [Design Constraints](#_2et92p0)

The design considerations take into account the aforementioned requirements. The game must be compatible with multiple platforms, and Apple has stringent guidelines regarding the programming languages permitted for application development (IOS App Dev Tutorials, n.d.). As the current version of the app is available exclusively for Android, it may necessitate reprogramming using the Swift programming language to enable functionality on Apple mobile devices. To ensure consistent gameplay across Windows machines, Linux distributions, and Apple machines, suitable interfaces must be established. Alternatively, the implementation of a software integration system that combines code to enable execution on different operating platforms could offer significant advantages.

## [System Architecture View](#_ilbxbyevv6b6)

Providing an overview of the system's structure and organization, it is crucial to assess the interactions between different components and platforms to achieve the ultimate objective of delivering an enjoyable gaming experience. To accomplish this, careful consideration should be given to the following aspects:

1. Components:
   1. How will the user interface effectively present the game? What business logic algorithms are involved? How will users communicate with the hosting/facilitating nodes? What measures are in place to secure information access and dissemination?
2. Platform-specific / Cross-platform architecture:
   1. How will the user interface, data access, algorithm implementation, and other programmatic aspects be addressed? Can duplication be minimized for specific platforms? Which platform-specific APIs are essential for the project?
3. Communication:
   1. How will data exchange occur between system nodes across diverse platforms?
4. Data sync:
   1. What mechanisms will ensure proper data synchronization? How will system requests for data sharing and feedback be prioritized?
5. Security / Access Control:
   1. How will user authentication and authorization be handled? What data should be public or private, and how will private data be protected?
6. Scalability:
   1. How will the application handle fluctuations in the number of system users? How will server loads be balanced? What caching strategies will be employed for data management?

While this list does not encompass all considerations, these areas of focus provide valuable insights into the communication between the system structure and its users during the project's development phases. Such understanding enables project leaders and stakeholders to address potential concerns in the planning stage.

## [Domain Model](#_8h2ehzxfam4o)

In the domain model, two principles, inheritance and composition, play significant roles (Freeman & Robson, 2020, p. 23). Within the com.gamingroom package, the Game, Team, and Player classes demonstrate an inheritance relationship with the Entity class. This relationship allows the Game/Team/Player classes to inherit attributes from the Entity class, reducing code duplication and facilitating future program modifications. The "is-a" test confirms this relationship, as the game, teams, and players exhibit the attributes and behaviors of the Entity class.

A composition relationship (ibid, 2020, p. 23) is evident between the GameService class and the subsequent classes in the UML diagram. This implies that a GameService "has-a" game, a game "has-a" team, and a team "has-a" player. Composition involves objects being composed of other objects during creation, resulting in a more complex object with additional functionality. The diagram shows that a game service can possess zero to many games, a game can contain zero to many teams, and a team can comprise zero to many players.

The final relationship is a simple association between the ProgramDriver class and the SingletonTester class. The ProgramDriver class holds the main method, serving as the entry point for a Java program. It utilizes the SingletonTester class to test the implementation of the Singleton Design pattern in the program. The Singleton Pattern ensures that a class has only one instance at a time and enables other classes to access that instance (Freeman & Robson, 2020, p. 177).

**"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** | Mac has superb terminal access and command structure. It is easily integrated with other Apple products and is popular for niche users, but has lower scalability and higher cost when compared to Linux or Windows. Mac does not easily integrate with other operating systems.  Mac does offer server-based deployment methods for a website to be hosted. That said, these tools have low adoption and use when compared to Linux or Windows based systems. Since the discontinuation the macOS Server software in 2022 (Apple Support, n.d.) there are no specific licensing costs associated with Apple servers for traditional server functionalities as these changes are now baked into the MacOS operating system. | Linux stands as the prevailing industry standard for hosting software applications on the server side. It combines a wealth of powerful terminal commands with cost-effective, open-source software implementation. However, Linux has a steep learning curve and a fragmented distribution set. Due to its open-source nature, vendor support for Linux is notably limited.  Linux is open source, and does not have a direct licensing cost for clients. While Linux server software is mostly free, there may be costs for enterprise-level services that require maintenance and support. Still, this cost is lower than Apple and the options are far greater. | When it comes to server-side hosting, Windows combines positive aspects of both Mac and Linux but doesn't surpass either. It boasts great vendor support, a robust PowerShell ecosystem, and great integration with hardware and software for hosting purposes. Although the learning curve is less steep than Linux, the Windows ecosystem tends to be more resource-intensive than either Linux or Mac.  Windows is not open source for enterprise server deployment and while there are options, there are proportionate costs incurred. Windows Server has varying pricing structures depending on project needs, and offers cloud options through Azure. | Mobile systems exhibit a significant degree of portability and widespread usage, making them more popular and cost-effective compared to non-mobile systems. However, one area where mobile systems fall short is their security, especially when compared to other dedicated systems that are hardwired and not reliant on mobility.  Mobile operating systems like Android and iOS are typically not licensed directly to clients for server-side usage.  When it comes to server-side usage, clients typically rely on server operating systems such as Linux or Windows. These server operating systems are responsible for handling the backend infrastructure and hosting the necessary software and services to support mobile applications. |
| **Client Side** | The cost of software development tends to be higher when working with Apple products due to the associated hardware expenses. While the time required for Mac-specific development is relatively low, it increases when considering the additional time needed for other concurrent operating platforms. However, using cross-platform development frameworks can alleviate this time constraint.  This device needs to interface with Chome, Edge, Safari, and more to facilitate cross-browser compatibility. Responsive web design will allow various screen sizes to all use the app. Testing, quality assurance, Apple’s proprietary APIs, and more will require special considerations when compared to a more ubiquitous Windows or Android OS. | Linux offers cost benefits due to its open-source nature, resulting in lower expenses. Moreover, Linux is compatible with a wide range of machines, ensuring platform independence. The time requirements for Linux are low, thanks to its high customizability and extensive documentation resources, coupled with strong community support. Moreover, Linux imposes relatively lower expertise requirements compared to other systems because there is a large pool of developers proficient in commonly-used Linux programming languages and well-versed in bash server commands.  As a point of consideration, Linux has a low adoption rate for client-side computing. | Windows proves to be cost-effective due to its dominant position in the industry, resulting in readily available server equipment. It offers a wide array of software systems and comprehensive documentation. While project complexity can influence development time, the availability of frameworks and integrated development environments (IDEs) helps mitigate this aspect. Also, there is a large talent pool of Windows-related programming experts, providing businesses with ample choices when it comes to expertise selection.  Windows has the largest adoption rate for client-side desktop OS, while it does share all the considerations laid out with macOS and Linux. | The cost, time, and expertise aspects of client-side software development are heavily influenced by the decision to adopt either a cross-platform development approach or to split the development between Android and iOS platforms. In the case of separation, iOS development inherits characteristics similar to Mac's, such as high cost, a higher expertise requirement due to Apple's specialized nature, and potentially lower development time. On the other hand, Android development shares some attributes with Windows, including the need for developers to understand the Android SDK, as well as other development tools, languages, and platform-specific APIs.  Client-side adoption for iOS/Android is extremely high, with ample APIs available for cross-platform or individual OS development. |
| **Development Tools** | The development tools used for Mac exhibit similarities to those available for Linux and Windows, such as Visual Studio Code, Sublime, Vim, and others. However, Mac-specific proprietary tools like Xcode are also utilized. Programming on Mac entails proficiency in the Swift programming language, along with knowledge of HTML, CSS, and JavaScript for front-end development. Moreover, Mac machines support various general-purpose languages, including Java (via the JVM), Python, and many more.  While tools like Xcode are free to use for Apple Development, deployment to the App Store requires enrollment in the Apple Developer Program [fee incurred]. This cost expands or contracts depending on the type of user. | Linux development tools encompass a range of text editors and integrated development environments (IDEs) such as Visual Studio Code, Vim, Sublime, and others. These tools facilitate the programming process on the Linux platform. For front-end development, the prevalent programming languages are HTML, CSS, JavaScript, and Python. On the server-side, developers commonly utilize languages like Java, C++, PHP, and JavaScript.  In terms of licensing costs for Linux development tools, it's essential to highlight that Linux itself is an open-source operating system, which means it is freely available for users to download, use, and modify. | Windows development tools offer a wide range of options to support developers in creating software applications. Notable IDEs such as Jetbrains' IntelliJ IDEA, Visual Studio, Visual Studio Code, as well as popular text editors like Vim and Sublime, are available for Windows users. Front-end development utilizes languages like HTML, CSS, and JavaScript, alongside Python and more. Back-end development uses general-purpose languages such as C++, Java, PHP, and more. Tools like Visual Studio and IntelliJ IDEA pro incur licensing costs for enterprise development augmentation, but there is no need to enroll in a special developer program like Apple requires. | While iOS developers rely on Apple’s Xcode, Swift UI, and UIKit to design, build, and test iOS applications, Android developers utilize the Android SDK and Google development tooling options. Apple’s Swift UI and UIKit provide frameworks and libraries that dovetail nicely with iOS devices, and should be considered if mobile deployment is chosen. The Android SDK includes Android Studio, which is the primary Integrated Development Environment (IDE) for developing Android apps.  The Android SDK and the Xcode development tools are both free, but Apple does still require those who deploy to the App Store to enroll in their Apple Developer Program. |

## 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**: In terms of the operating platform, it is recommended that The Gaming Room commences its development process with Windows devices. Windows offers a well-balanced approach to cost, time, and expertise requirements for this project. While there are licensing costs associated with various server and client-side development tools, Windows provides the most comprehensive range of options when it comes to developing for multiple operating platforms.

Moreover, Windows offers extensive support for game development, featuring robust and well-documented user libraries and frameworks like libGDX (Giuseppe et al., 2016, p. 7) and jMonkey3 (Slant, 2023). Leveraging these tools can greatly enhance the rendering and performance of the Draw It or Lose It game application across diverse devices. Furthermore, the widespread popularity of Windows as a desktop operating system ensures a significant user base, thereby expanding the potential reach of the game.

To maximize the game's reach and accessibility, it is essential to consider platform compatibility. By incorporating cross-platform development frameworks and leveraging the capabilities of web technologies like HTML5, CSS, and JavaScript, The Gaming Room can extend its game application to other operating systems such as macOS, Linux, and mobile platforms like iOS and Android. This approach allows for broader user engagement and ensures that the game can be enjoyed across a wide range of devices, enhancing its overall success and user satisfaction.

1. **Operating Systems Architectures**: The kernel serves as the fundamental pillar of the Windows architecture, as elucidated by Hudek et al. (2021). It encompasses a core set of programs that facilitate the essential functionalities of the operating system. Windows operating systems, including Windows 7, 8, 10, and 11, leverage the Windows API, as explained by Marcho (2019). This API enables a wide range of operations and provides a graphical user interface while maintaining compatibility with earlier versions, fostering a sense of continuity.

A notable characteristic of Windows architecture is its unwavering dedication to backward compatibility. Windows operating systems strive to ensure that applications developed for older versions can seamlessly run on newer versions with minimal adjustments. This commitment allows for a smooth transition for both users and developers, reducing disruptions and providing a consistent experience across various Windows platforms.

In addition to its kernel and backward compatibility, Windows architecture benefits from extensive support for game development. User libraries and frameworks like libGDX (ibid 2016) and jMonkey3 (ibid, 2023) greatly enhance the rendering and performance of game applications, catering to the diverse array of devices on which Windows is widely utilized. These resources contribute to the popularity and versatility of Windows as a platform for game development, expanding the potential reach and impact of game applications developed within its ecosystem.

1. **Storage Management**: Windows presents a wide range of capabilities for managing storage, encompassing functionalities such as disk partitioning, file systems, disk management consoles, disk cleanup tools, and an embedded auto-manager called Storage Sense (Microsoft, n.d.). Storage Sense is seamlessly integrated into Windows 10 and subsequent versions, providing automated management of storage resources.

By capitalizing on the diverse storage management capabilities offered by Windows, The Gaming Room can effectively oversee the storage requirements of the Draw It or Lose It game application. These features facilitate efficient organization, allocation, and cleanup of storage space, ensuring optimal performance and storage utilization. Notably, Storage Sense automates various storage management tasks, reducing the need for manual intervention and providing a hassle-free storage management experience.

Furthermore, Windows also provides developer-friendly tools and APIs for storage management, allowing The Gaming Room to implement custom storage solutions tailored to the specific requirements of the game application. Leveraging these resources empowers developers to optimize storage usage, enhance data access efficiency, and provide a seamless and responsive gaming experience for players.

1. **Memory Management**: Windows incorporates various memory allocation techniques, including swapping and paging. As outlined by Silberschatz et al. (2009), swapping involves moving processes in and out of memory as needed for execution, while paging enables infrequently used processes to be transferred to a backlog when system memory is limited. These techniques effectively prevent fragmentation and potential data corruption.

Swapping, as described by Rouse (2014), is a memory management approach employed by the operating system. It entails transferring processes between the main memory (RAM) and secondary storage (typically the hard disk) as required. When a process is not actively executing, it can be swapped out of memory and stored on the disk, freeing up space for other processes. Subsequently, when the process needs to be executed again, it is swapped back into memory. Swapping optimizes memory utilization by enabling the accommodation of more processes than the physical memory can accommodate simultaneously.

Finally, Windows incorporates advanced memory management features and algorithms to optimize system performance. It utilizes techniques like demand paging, which loads only necessary parts of a program into memory, and memory compression, which compresses memory pages to conserve space. These mechanisms help improve overall system responsiveness and efficiency, ensuring smooth execution of the Draw It or Lose It game application even in memory-constrained scenarios.

1. **Distributed Systems and Networks**: For the development of this game, JetBrains' IntelliJ IDEA is the recommended integrated development environment (IDE). This powerful IDE offers extensive support for application development across multiple platforms, including Windows, Mac, Linux, and more. It also facilitates cross-platform development approaches similar to those utilized by React-Native (React, n.d.), and comes with tooling to integrate with various version control systems. Leveraging such capabilities allows for the deployment of applications on diverse cloud servers, harnessing scalable network services to enhance performance and flexibility.

When designing the game application in a web-based distributed environment, it is crucial to consider the dependencies between the system components and the underlying network connectivity. The application should be resilient to network outages or fluctuations, ensuring a seamless gaming experience even in challenging network conditions. Incorporating error handling mechanisms, offline data synchronization, and intelligent conflict resolution strategies can significantly contribute to maintaining uninterrupted gameplay and user satisfaction.

Furthermore, security measures [see below] should be prioritized to safeguard sensitive user information and protect against potential threats. Implementing secure network communication protocols, encryption algorithms, and authentication mechanisms will help ensure the confidentiality and integrity of user data. By adhering to industry best practices and considering the implications of security within the distributed environment, The Gaming Room can establish trust and confidence among its users while maintaining the integrity of the game application.

1. **Security**: In order to uphold system security, users need to undergo appropriate authentication and authorization processes. Compliance with user data privacy laws - which can vary internationally - is of utmost importance. Hence, it is crucial to validate the legal requirements and potentially seek external legal consultation. By embracing cloud storage and distribution strategies provided by reputable cloud vendors, much of the effort associated with constructing software security from scratch can be…scratched. These cloud-based solutions offer robust security measures and ease some of the burdens associated with reinventing the wheel by implementing vertically integrated comprehensive security measures.

By implementing a thorough security strategy that encompasses proper authentication and authorization processes, adherence to data privacy laws, and utilization of secure cloud storage and distribution strategies, The Gaming Room can foster user confidence and safeguard user information across multiple platforms.

Moreover, it is imperative to regularly update and patch the game application to address any potential security vulnerabilities that may arise over time. By staying vigilant and proactive in addressing security concerns, The Gaming Room can stay one step ahead of potential threats and ensure the long-term security and integrity of the Draw It or Lose It game application.

**References**

Apple Support. (n.d.). *About macOS Server 5.71 and later.* Apple. https://support.apple.com/en-us/HT208312.

Freeman, E., & Robson, E. (2020). *Head first design patterns*. O’Reilly Media.

Guiseppe, S. D., Kruhlmann, A., & Rijnswou, E.V. (2016). *Building a 3D game with libGDX.* Packt Publishing. https://eds-p-ebscohost-com.ezproxy.snhu.edu/eds/ebookviewer/ebook/bmxlYmtfXzEzNDM4ODVfX0FO0?sid=1600627b-c1b3-42d6-af61-33564cb02ee5@redis&vid=1&format=EB&rid=1.

Hudek, T., Seattle, E., & Sherer, T. (December 14, 2021). *Windows kernel-mode kernel library*. Microsoft. https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/windows-kernel-mode-kernel-library.

IOS App Dev Tutorials. (n.d.). *Develop apps for iOS*. Apple. https://developer.apple.com/tutorials/app-dev-training.

Marcho, C. (April 10, 2019). *Windows architecture – The basics*. Microsoft. https://techcommunity.microsoft.com/t5/ask-the-performance-team/windows-architecture-the-basics/ba-p/372345.

Microsoft Staff. (n.d.). *Manage drive space with Storage Sense.* Microsoft. https://support.microsoft.com/en-us/windows/manage-drive-space-with-storage-sense-654f6ada-7bfc-45e5-966b-e24aded96ad5.

React Native. (n.d.). *Getting started: React Native.* React. https://reactnative.dev/docs/getting-started.

Rouse, M. (October 10, 2014) *Memory Swapping.* Techopedia. https://www.techopedia.com/definition/30467/memory-swapping.

Silberschatz, A., Galvin, P. B., & Gagne, G. (2009). *Operating system concepts, 8th edition.* Wiley. https://learning.oreilly.com/library/view/operating-system-concepts/9780470128725/silb\_9780470128725\_oeb\_c08\_r1.html#h4.

Slant. (2023). *What are the best Java game engines?* Slant. https://www.slant.co/topics/21245/~java-game-engines.