

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 |
| --- | --- | --- | --- |
| 1.0 | 11/9/23 | Jordan Mitchell | Initial prototype design |
| 2.0 | 11/24/23 | Jordan Mitchell | Second design draft |
| 3.0 | 12/7/23 | Jordan Mitchell | 3rd and final draft |

**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)

CTS was approached by The Gaming Room to create a web-based version of their currently existing Android game Draw It or Lose It. The game allows for multiple teams each with multiple players. For proper functionality, each team name and game-instance must be unique. The game will render images from a large library of stock drawings. The Gaming Room’s team are not able to set up the environment on their own.

## Requirements

* *A specific game-instance must have the ability to support one or multiple teams.*
* *Each team must be able to support multiple players.*
* *The names of each team, player, and game-instance must be unique, and the game must allow for users to check the availability of names.*
* *Only one instance of a game can exist at any given time. Unique identifiers must be created for each instance, player, and game.*

*To meet these requirements, a singleton creation pattern will be utilized to prevent multiple instances of a game, while an iterator pattern will be utilized to prevent identical team and player names.*

## [Design Constraints](#_2et92p0)

-Since The Gaming Room would like to serve multiple platforms, the API will need to be tailored to work on all 3 platforms

-Web, Android, and iOS all have separate software development kits (SDKs)

-Team, player, and game names must all be unique

-API must be able to allow one or more teams from each of the platforms

## [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 ProgramDriver class contains the main method. There is direct association between ProgramDriver and SingletonTester, which assesses if there is already an instance of GameService. The creation is done through the GameService class, following a singleton pattern, ensuring only one GameService class exists at any time.

GameService sets its constructor to private to prevent outside creation of repeat instances. To instantiate GameService, getInstance() must be used to check if GameService is already present in the memory.

When GameService is active, the driver uses addGame() which utilizes the iterator pattern. This ensures there are no repeat names for any game instances. After creating a new game, the game is added to the list of games.

When the game instance has been created, teams are added through addTeam(), which similarly utilizes the iterator pattern to prevent repeat team names. After adding a new team, the team is added to the list of teams.

The process continues, where once a team is created players can be added via addPlayer(). This method also uses the iterator pattern to ensure there are no repeat player names.

Entity is the parent class to the Game, Team, and Player classes. Game, Team, and Player all inherit Entity’s required attributes. A Team can have a Player, but a Player can’t have a Team. Similarly, a Game can have a team, but a Team can’t have a Game, and so on.

The UML diagram shows many aspects of object-oriented programming (OOP). Inheritance and polymorphism are present and used as extensions of the Entity parent class. Abstraction and encapsulation are used in the method to add teams/games/players as those objects cannot be created through accessing the constructor.

**"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 can be utilized to host servers. The concerns would be the cost of the licensing as well as purchasing hardware (MacBooks)to develop with.   OS X Server is no longer available as Apple has phased it out. The cost was $500. | Linux is completely capable of web-based hosting. Linux is free and open source. Does not require as many resources compared to Windows. Licensing and maintenance costs would be cheaper than an OS like Windows. | Windows has many web-based hosting services available. They are easily set up and secure. A concern would be the high cost of licensing especially at it is typically per user. Windows server licenses can cost upwards of $6000. | Mobile devices, while capable, are not optimal. They are lacking in power to perform on the high-end. It’s possible to utilize them for development but should not be the first choice for web-hosting. |
| **Client Side** | Mac is very capable with easy-to-use software development kits. The drawback is that you must have native hardware (macs) to develop, meaning more cost and requiring a developer experienced in swift.   Per [Statista](https://www.statista.com/statistics/218089/global-market-share-of-windows-7/#:~:text=Microsoft's%20Windows%20was%20the%20dominant,share%20of%20around%2070%20percent.) Windows dominates the market with a 70% usage. | The tradeoff for Linux would be the extensive development time. A requirement would be a developer comfortable utilizing Linux as not all are comfortable doing so. | Developers will require platform-specific expertise. A consistent user experience, maximizing code reusability, and addressing compatibility changes are very important. identify a good framework for security and capability purposes. Windows being the premier operating system used by 70% of users means it is the better choice business-wise. | You must have developers experienced in creating apps. User experience and interface are essential and must be handled differently than the web.  Since iOS is mainly Swift based, the needs are the same as Mac’s including the hardware requirements. |
| **Development Tools** | Apple’s products come with an integrated IDE: Xcode. This can be used to code in swift.  Xcode is listed as costing $99 PER developer. | Python automatically comes with Python pre-installed on most distributions of Linux. PyCharm or IntelliJ can be used to code.  Python IDEs are often free, such as Pycharm and Notepad++.  For C++ development, the best option would be Eclipse as it's free, and capable. | Most Windows programs are typically written in Java, C, and C++. PyCharm, Visual Studio Code, and IntelliJ are all options to code windows applications.   The cost of Visual Studio starts at $45 and can increase to around $250 per user per year, depending on the features and plan purchased. | Mobile devices are more limited in their options. For IOS, Xcode can be utilized to develop in swift. Alternatively, the app can be developed with Unity, which uses C++, before finally being converted to either Android or IOS. However, a Mac will still be required to convert to iOS. For developing an Android app, Android Studio can be utilized, but a developer with experience with this is a requirement.  Xcode is listed as costing $99 PER developer. |

## 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 most appropriate operating platform that will allow them to expand to other environments in my opinion would be a Linux based server platform like Ubuntu. Linux is open source meaning licensing costs are reduced and there is a high level of community support. Linux offers strong security, stability, and flexibility. The data centers do not have limited access as can be the case with other platforms like Windows. Another benefit is its agnostic coding capabilities, meaning the front end can be developed in the preferred language of the platform independent of the back end. For example, Java can be used for Android, Swift can be used for iOS, etc. The drawback being not as many developers may be comfortable or experienced with Linux compared to Windows, the most used OS in the world. The cost of training or hiring experienced developers should be considered.
2. **Operating Systems Architectures**: Linux functions through a monolithic architecture. This means the entire system is operating in kernel mode. This enables high performance through a faster system call. The single kernel directly interacts with the hardware, providing control over various resources and an efficient performance. Linux is capable of supporting various hardware and is scalable.   
   The proposed architecture will segregate the backend server for game management and utilize a client-based approach to render the frontend. This will optimize resource utilization by pushing the rendering tasks to the client side, reducing data center costs. This also has the potential benefit of improving gameplay by isolating it from any network issues.
3. **Storage Management**: Should decide between hard disk drives and solid-state drives to meet the performance requirements of the application. This is important especially with regards to caching and rendering on the client side. On the server side, cloud tools can be used to add flexibility. A recommended system would be a distributed file system (DFS) like Ceph or GlusterFS. These systems can be scaled, are tolerant of faults, and can handle large amounts of data like the game’s image files. A DFS ensures data redundancy making expansion and availability easy across multiple servers.
4. **Memory Management**: Linux has virtual memory management techniques, including memory mapping and demand paging. These techniques help to optimize memory usage. These efficiently manage memory allocation and sharing amongst the processes to prevent memory leaks and enhance overall performance. Linux utilizes page cache for data that is stored in the main memory. Demand paging works by not loading unused pages into the memory. There is also the last recently used algorithm that replaces the most recent page whenever a page fault occurs. This means the page that has not been utilized for the longest time in memory, it is replaced by a new one. For client-side rendering, there will be minimum RAM requirements on the server. If you are utilizing an architecture with microservices and containers, the cost gradually increases alongside the number of users.  
     
     
   Memory management on iOS devices has stricter guidelines due to Apple’s memory and sandbox restrictions. Many Apple devices have more limited RAM compared to PCs, meaning efficient memory use is vital. Apple’s iOS has automatic memory management through Automatic Reference Counting (ARC), which manages memory allocation and deallocation for objects. Developers should look to minimize memory leaks and optimize usage of resources.  
     
   Android devices have significantly more variation in terms of hardware capabilities and memory availability. Android's memory management relies on the Java Virtual Machine (JVM) for apps developed in Java or Kotlin. Android also has garbage collection, which manages memory periodically by reclaiming memory from objects that are no longer needed. Developers should consider efficient bitmap usage and minimize memory leaks. Tools like Android Profiler can be used to optimize and analyze memory use.
5. **Distributed Systems and Networks**: For game communication between various platforms, you can use distributed software and network connectivity. Ensure components handle data compatibility and consider network reliability and redundancy to deal with outages to create a smooth cross-platform gaming experience. A RESTful API based architecture can be implemented to communicate across platforms. Load balancers and distributed caching mechanisms like Redis can be used for management of sessions and faster retrieval of data. Uptime and outage prevention are a driving force behind the development of applications in cloud architectures. Cloud providers can shift and create services amongst deployments to prevent outages and excessive downtime on a large scale. Usage of RESTful API allows for client to server communication transparency to whichever frontend, either iOS, Windows, or Android.
6. **Security**: To safeguard user information across different platforms, you can implement strong encryption and authentication methods in the software. Linux-based systems offer strong security through frequent patches, and a strong permission-based access control. Encryption protocols like SSL and TLS will help secure data transfer between the platforms. Implementation of role-based access control (RBAC) will manage user permissions effectively. This means the interface for an entitlement will be created for effective administration of the roles and accounts.   
   Users should be limited to minimal privilege, only allowing them access to game controls like creation of teams and games, naming conventions, etc. If the Gaming Room wishes, user privilege can include a hierarchy of members that include a game host or team leader with the power to edit teams at will. **No users should be allowed admin privileges**.  
   A firewall should be included so as to best adhere to industry standards.

Resources:  
Baeldung. "Monolithic Kernel in Linux." <https://www.baeldung.com/linux/monolithic-kernel>.

TechTarget. "Distributed File System (DFS) - Definition." <https://www.techtarget.com/searchstorage/definition/distributed-file-system-DFS>.

Cloudflare. "What is Caching?" <https://www.cloudflare.com/learning/cdn/what-is-caching/>.

Thomas-Krenn. "Linux Page Cache Basics." <https://www.thomas-krenn.com/en/wiki/Linux_Page_Cache_Basics>.

O'Reilly. "Understanding the Linux Kernel." <https://www.oreilly.com/library/view/understanding-the-linux/0596005652/ch15s01.html>.

UC Berkeley EECS. "CS162 - Operating Systems and System Programming." <https://inst.eecs.berkeley.edu/~cs162/su21/static/sections/section8-sol.pdf>.

Educative. "What is the Least Recently Used (LRU) Page Replacement Algorithm?" <https://www.educative.io/answers/what-is-the-least-recently-used-page-replacement-algorithm>.

Apple Developer. "Memory Management in Cocoa." <https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MemoryManagement.html>.

TutorialsPoint. "iOS Memory Management." <https://www.tutorialspoint.com/ios/ios_memory_management.htm>.

Swift.org. "Automatic Reference Counting in Swift." <https://docs.swift.org/swift-book/documentation/the-swift-programming-language/automaticreferencecounting/>.

LogRocket Blog. "Complete Guide to Android Profiling." <https://blog.logrocket.com/complete-guide-android-profiling/>.

Imperva. "Role-Based Access Control (RBAC)." <https://www.imperva.com/learn/data-security/role-based-access-control-rbac/#:~:text=Role%2Dbased%20access%20control%20(RBAC)%2C%20also%20known%20as,enable%20access%20to%20authorized%20users>.

Digital Guardian. "What Is Role-Based Access Control (RBAC)? Examples, Benefits, and More." <https://www.digitalguardian.com/blog/what-role-based-access-control-rbac-examples-benefits-and-more>.

TechTarget. "IT Security Frameworks and Standards: Choosing the Right One." <https://www.techtarget.com/searchsecurity/tip/IT-security-frameworks-and-standards-Choosing-the-right-one>.

US Government Publishing Office. "IT Security Frameworks and Standards (PDF)." <https://www.govinfo.gov/content/pkg/GOVPUB-C13-f52fdee3827e2f5d903fa8b4b66d4855/pdf/GOVPUB-C13-f52fdee3827e2f5d903fa8b4b66d4855.pdf>.  
  
ESDS Blog. "Battle of Web Hosting Platforms: Mac vs Linux vs Windows." <https://www.esds.co.in/blog/battle-of-web-hosting-platforms-mac-vs-linux-vs-windows/#sthash.hDn38BJG.dpbs>.

GeeksforGeeks. "Top Programming Languages for Android App Development." <https://www.geeksforgeeks.org/top-programming-languages-for-android-app-development/>.

Linux.com. "Overview of Linux Kernel Security Features." <https://www.linux.com/training-tutorials/overview-linux-kernel-security-features/>.

Vault Networks. "Pros and Cons of In-House Cloud Management vs Outsourcing." <https://www.vaultnetworks.com/pros-and-cons-of-in-house-cloud-management-vs-outsourcing/>.

Computerworld. "Why Linux is Better than Windows or MacOS for Security." <https://www.computerworld.com/article/3252823/why-linux-is-better-than-windows-or-macos-for-security.html>.

AWS Documentation. "What is a RESTful API?" <https://aws.amazon.com/what-is/restful-api/#:~:text=RESTful%20API%20is%20an%20interface,applications%20to%20perform%20various%20tasks>.

RESTfulAPI.net. “What is Rest?” <https://restfulapi.net/>