

<N 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 | <07/27/24> | <Allen Falcon> | < This revision introduces an `Entity` class to encapsulate common attributes (`id` and `name`) and refactors the `Game`, `Team`, and `Player` classes to inherit from `Entity`. The `GameService` class now follows the singleton pattern to ensure only one instance manages game data, with unique identifiers starting from 1 for games, teams, and players. The iterator pattern is used to enforce unique names for these entities. Additionally, sample games, teams, and players were created in `ProgramDriver` for testing, and the `SingletonTester` class verifies the singleton behavior of `GameService`. > |
| 2.0 | 8/8/24 | Allen Falcon | Check Diffrent platforms |
| 3.0 | 8/18/24 | Allen Falcon | Updated recom |

**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 has tasked Creative Technology Solutions with the development of a webbased version of their existing Android game, Draw It or Lose It. This game, inspired by the 1980s television game Win, Lose or Draw, requires players to guess images drawn from a large library of stock drawings within a set time limit.

Our solution involves creating a distributed web-based game application that supports multiple teams, unique game and team names, and ensures only one instance of the game runs in memory at any given time. This will be achieved through unique identifiers for games, teams, and players. We will document the development process by creating a comprehensive software design document that outlines the necessary design constraints and domain model for the game application.>

## Requirements

*<*

The game must support one or more teams.

Each team can have multiple players.

Game and team names must be unique to prevent duplication.

Only one instance of the game can exist in memory at any given time, necessitating unique identifiers for each game, team, and player. *>*

## [Design Constraints](#_2et92p0)

< The application must efficiently handle multiple teams and players simultaneously, ensuring smooth gameplay without lag or downtime

Implementing a system that allows only one instance of the game in memory at a time, requiring robust concurrency control and synchronization mechanisms.

Ensuring uniqueness for game, team, and player names across the platform, which involves implementing a reliablely naming and validation system.

Ensuring the application functions seamlessly across different web browsers and devices, which requires thorough testing and possible adaptation of the user interface and functionality.

>

## [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 Gaming Room's game application illustrates a well-structured domain model, demonstrating key object-oriented programming principles. The `Entity` class acts as a base, encapsulating common attributes (`id` and `name`) and methods (`getId`, `getName`, and `toString`), which are inherited by `Game`, `Team`, and `Player` classes. This inheritance promotes code reuse and maintainability. The `GameService` class employs the singleton pattern, ensuring only one instance manages the game's state, while its composition relationships with `Game`, `Team`, and `Player` classes reflect the hierarchical structure of the game. Encapsulation, polymorphism through overridden `toString` methods, and clear associations between classes enhance the modularity and scalability of the application, efficiently meeting the 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** | < Mac offers stability and security but is costly and less common for servers. > | < Linux is cost-effective, secure, and highly customizable but requires expertise. > | < Windows is user-friendly and integrates well with other Microsoft products but is expensive and resource-intensive. > | Limited for hosting; better suited as client devices interacting with web servers. > |
| **Client Side** | < High development costs and need for macOS expertise; ensures seamless experience for Apple users. > | <  Cost-effective and flexible; requires expertise in various distributions. > | < Straightforward with powerful tools like Visual Studio; broad user base but has licensing costs..> | < Requires expertise in iOS and Android; ensures wide reach but increases development time and costs. > |
| **Development Tools** | <I Tools: Xcode, Sublime Text, Atom. Languages: Swift, Objective-C. > | <I GCC, Clang, Eclipse. Languages: Python, Perl, Java. > | <\ Visual Studio, PowerShell. Languages: C#, C++, .NET. > | < Android Studio, Xcode. Languages: Java, Kotlin, Swift, Objective-C..> |

#### Server Side

* **Mac**: Mac offers stability and security but is costly and less common for servers. It can be used for server-side hosting, but the **high cost** of hardware and software may not justify the investment compared to other platforms.
* **Linux**: Linux is cost-effective, secure, and highly customizable but require expertise. It is the preferred choice fo**r server-side** hosting due to its flexibility, stability, and **support for various server-based deployments.**
* **Windows**: Windows is **user-friendly** and integrates well with other Microsoft products but is expensive, resource-intensive, and has traditionally had **more security vulnerabilities** compared to other platforms like Linux. It supports serverbased deployments effectively but involves higher licensing costs and **may require additional security measures** to protect against common threats.
* **Mobile Devices**: **Limited for hosting**; better suited as client devices interacting with web servers. Mobile devices are not typically used for server-side hosting due to their hardware constraints and **limited processing power.**

#### Client Side

* **Mac**: High development costs and need for macOS expertise; ensures seamless experience for Apple users. Developing for macOS requires familiarity with Apple's development tools and guidelines, which can be costly but provides a polished user experience for Mac users.
* **Linux**: Cost-effective and flexible; requires expertise in various distributions. Developing for Linux is generally less expensive and ffers great flexibility, but it requires knowledge of different Linux distributions and their unique style.
* **Windows**: Straightforward, with powerful tools like Visual Studio; broad user base but has licensing costs. Development for Windows is streamlined with strong tools but incurs licensing fees, which can add to the overall cost.
* **Mobile Devices**: Requires knowledge in iOS and Android; ensures wide reach but increases development time and costs. Supporting both iOS and Android means handling two distinct ecosystems, which can be **time-consuming** and costly but necessary for broad user reach.

#### Development Tools

* **Mac**: Tools: Xcode, Sublime Text, Atom. Languages: Swift, Objective-C. Developing on Mac requires specific tools and languages that are tailored to Apple's ecosystem, which can incur **additional costs** for licenses and training.
* **Linux**: Tools: GCC, Clang, Eclipse. Languages: Python, Perl, Java. Linux development is versatile with many **open-source tools available**, making it cost-effective, but it requires proficiency in multiple languages and tools.
* **Windows**: Tools: Visual Studio, PowerShell. Languages: C#, C++, .NET. Windows offers a comprehensive development environment with Visual Studio but involves licensing costs and a steep learning curve for certain technologies.
* **Mobile Devices**: Tools: Android Studio, Xcode. Languages: Java, Kotlin, Swift, Objective-C. Developing for mobile platforms needs understating in multiple development environments and languages, which can increase complexity and costs.

### Recommendations

Based on the evaluation, the following recommendations are made for The Gaming Room :3

* **Operating Platform**: Use Linux for its stability, security, flexibility, and cost-effectiveness.
* **Operating Systems Architectures**: Linux's modular and customizable architecture meets various server needs and can handle a wide range of server-side tasks efficiently.
* **Storage Management**: Combine fast local SSD storage with scalable cloud storage solutions to ensure quick acces times and high scalability.
* **Memory Management**: Linux efficiently handles memory with tools like top and vmstat, making it a ok choice for managing resources.
* **Distributed Systems and Networks**: Use RESTful APIs and Sockets for communication, and Docker and Kubernetes for scalability and orchestration.
* **Security**: Ensure robust security with SSL/TLS, firewalls, IDS, user authentication, and regular updates to protect the application and its users.

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:** Use Linux for its stability, security, flexibility, and cost-effectiveness.
2. **Operating Systems Architectures:** Linux's modular and customizable architecture meets various server needs.
3. **Storage Management:** Combine fast local SSD storage with scalable cloud storage solutions
4. **Memory Management:** Linux efficiently handles memory with tools like top and vmstat.
5. **Distributed Systems and Networks:** Use RESTful APIs and Sockets for communication, and Docker and Kubernetes for scalability.
6. **Security:** Ensure robust security with SSL/TLS, firewalls, IDS, user authentication (OAuth2), and regular updatess.

To help The Gaming Room expand Draw It or Lose It across different platforms, I recommend using Linux as the main operating system. Linux is flexible, secure, and efficient, making it a great choice for running the game on servers. The architecture of Linux includes the core system (kernel), user space where applications run, and a well-organized file system. This setup offers stability and security, which are crucial for keeping the game running smoothly and scaling as needed. Plus, since Linux is open-source, it’s cost-effective and can be customized to fit exactly what The Gaming Room needs. There’s also strong support from both the community and businesses, so we can be confident that Linux will continue to get updates and security improvements.

For managing storage, I suggest using Linux's Logical Volume Manager. LVM provides a flexible way to manage disk space, allowing the creation of logical volumes that can be resized on the fly without causing downtime. This is particularly important for Draw It or Lose It. where the game data, including assets and user-generated content, might grow over time. LVM also supports features like snapshotting, which can create point-in-time copies of the data. This is essential for quick backups and disaster recovery, ensuring that if something goes wrong, you can easily roll back to a previous state without losing much progress. Additionally, LVM can aggregate multiple physical disks into a single logical volume, improving performance and providing redundancy, which is crucial for maintaining high availability and reliability of the game’s data.

In addition to using LVM, leveraging cloud storage is highly recommended. Cloud storage offers scalability, allowing the storage capacity to grow as the game’s data needs increase, without the constraints of physical hardware. It also provides accessibility, ensuring that data can be accessed from anywhere, which is essential for a distributed game accessed by players from different devices and locations. Cloud storage is cost-effective, as you typically pay only for what you use, and it includes built-in backup and disaster recovery options, minimizing the risk of data loss. Services like Amazon S3 or Google Cloud Storage would be excellent choices for our game offering reliability, security, and easy integration with other cloud-based services.

When it comes to managing memory, Linux offers advanced techniques that ensure efficient use of system resources. One of the key techniques is virtual memory management, which allows the system to use disk space as an extension of RAM, effectively increasing the amount of memory available to applications. This is particularly useful for our game if the game needs more memory than what is physically available on the server. Linux also uses a technique called paging, where memory is divided into small blocks, or pages, and only the most frequently used pages are kept in physical RAM. This reduces the load on memory and ensures that critical game data is quickly accessible. Additionally, Linux provides caching mechanisms that store frequently accessed data in memory, reducing the need to repeatedly fetch the same data from disk, which can significantly speed up game performance. For example, frequently used game assets or player data can be cached in memory, ensuring that the game runs smoothly and with minimal latency.

For supporting a distributed system, where different parts of the game run on different servers, Linux is reliable and supports various networking protocols. This makes it easier to manage the game across multiple platforms, with load balancing, redundancy, and fault tolerance all built in. We can also secure communications between servers using tools like SSH and VPN, which are well-supported on Linux.

Security is a top priority for The Gaming Room, and Linux offers strong protection for user data. Linux provides robust authentication methods and supports encryption tools like GPG and SSL/TLS to keep data safe both when stored and during transmission. It also has powerful firewall tools like iptables and firewalld to protect against unauthorized access. Since Linux is open-source, we can customize the security features to meet the specific needs for us.

In summary, choosing Linux as the operating system for our requirements gives The Gaming Room a strong and scalable foundation. Linux’s architecture, efficient memory and storage management, including the use of cloud storage, support for distributed systems, and strong security features make it an ideal choice for expanding the game to different platforms while keeping it secure and performing well.