

<Name-of-Software-Application>

# **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 | 11/07/2023 | Vincent Noce | In the revised evaluation and recommendations, The Gaming Room's Draw It or Lose It application has been considered based on the scenario and requirements. |

**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)

Creating a draw-it-or-lose-it application for various platforms and devices is a challenge for the Gaming Room's Draw It or Lose It application. Linux-based server platforms are cost-effective and scalable, so we recommend them for this application. Use responsive design for client-side development to ensure a consistent user experience across platforms. To meet the client's goals, we focus on efficiency and cost-effectiveness.

## Requirements

An executive summary outlined the client's requirements, emphasizing platform compatibility, scalability, and security.

## [Design Constraints](#_2et92p0)

Game application development will be based on a web-based distributed environment, with consideration given to reducing network latency, maintaining data consistency, and maintaining seamless communication between servers.

## [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)

**Relationships**:

* **Inheritance**: **Player**, **Team**, and **Game** instances inherit attributes (id, name) and methods from Entity instances. This illustrates the principles of code reusability and abstraction in object-oriented design.
* **Composition**: Player and Team objects are listed in the Team and Game classes, indicating a composition relationship.
* **Singleton Pattern**: Only one instance of GameService is in memory for centralized game management, which is ensured by a singleton implementation.

**Object-Oriented Programming Principles**:

* **Inheritance**: It allows you to maintain a common structure for all Entity classes (Player, Team, Game) by promoting code reusability.
* **Composition**: The Team and Game classes combine simpler objects (players and teams, respectively) to create complex objects (teams and games, respectively).
* **Singleton Pattern**: By implementing GameService as a singleton, we ensure centralized control and efficiency of game management.

**"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

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 Mac is certainly a powerful platform, its commercial licensing fees may negatively impact overall expenses. In Serv-based deployments, Draw It or Lose It can be cost-effectively deployed using a Linux platform. | Linux, Mac, Windows, and mobile devices are all supported by Serv-based deployments. The open-source solution of Linux reduces licensing costs. Mac and Windows require commercial licenses, which may impact overall costs. Thousands of players must be accommodated simultaneously while taking scalability into consideration. | Commercial licensing may be required for Windows. Considering Linux's cost-effectiveness, it is recommended for server-side deployments. | Linux is recommended for server-side deployment. There is support for both iOS and Android platforms, so a wider audience can be reached. |
| **Client Side** | Client-side development can be conducted on Macs thanks to their stable architecture. An integrated user experience can be achieved through Xcode, Swift, HTML, CSS, and JavaScript development tools. | There are tests conducted to ensure compatibility; cross-browser compatibility is offered. By using responsive design, users can access content seamlessly across a variety of devices | Tools for developing websites, such as Visual Studio Code, HTML, CSS, and JavaScript. Licenses do not need to be purchased separately; Expertise across platforms. | In mobile development, Swift, Xcode, and responsive design are used. An overview of Android Studio and Java; responsive design considerations. |
| **Development Tools** | A key development tool for Mac is Xcode, which provides comprehensive support for developing iOS applications. | HTML, CSS, JavaScript are all supported by Xcode for iOS development. Collaboration with iOS developers; Xcode licensing costs. | Tools for development:  VS Code and VS Visual Studio. | JavaScript, HTML, and CSS in Visual Studio. Collaboration with Windows developers; Microsoft Visual Studio licensing costs. |

* *Server Side:* The open-source nature of Linux, cost-effectiveness, and ability to use Serv-based deployments make it an excellent choice for server-side deployments. Mac and Windows, requiring commercial licenses, may impact overall costs.
* *Client Side:* Testing is conducted to ensure cross-browser compatibility, and responsive design is utilized for seamless content access across various devices. Web-based environments require cross-platform development tools such as VS Code, HTML, CSS, and JavaScript. For mobile development, Swift, Xcode, Android Studio, Java, and responsive design considerations are highlighted.

## 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 cost-effectiveness and open-source nature of Linux make it the best server-side deployment choice for Draw It or Lose It.
2. **Operating Systems Architectures***:* Operating systems based on Linux are popular and suitable since they are stable and scalable.
3. **Storage Management***:* It is recommended that data integrity and availability be ensured through distributed file systems with redundancy.
4. **Memory Management***:* Software such as Draw It or Lose It benefits from Linux's efficient memory management techniques.
5. **Distributed Systems and Networks***:* Using RESTful APIs and microservice architectures, different platforms can communicate with each other, emphasizing the importance of redundancy and connectivity.
6. **Security***:* In order to protect user information across platforms, HTTPS and encryption protocols are recommended. Security software should also be audited and updated regularly.