

<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 | 01/25/25 | Ethan Compton | Updates were applied to the cover page, document revision history, executive summary, design constraints, system architecture view, domain model, and 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)

The Gaming Room project aims to develop a web-based game compatible with multiple platforms, expanding on the existing "Draw It or Lose It" game, which is currently limited to Android. The game involves multiple teams, each consisting of several players, competing over four one-minute rounds. During each round, a picture is selected from a library of images, and one team attempts to guess the image before time runs out. If the team fails to guess correctly, opposing team members have 15 seconds to provide an answer.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_2et92p0)

The Entity class acts as a common superclass that establishes a relationship among the Game, Team, and Player classes, enabling them to inherit or share information. In UML, this is shown as inheritance with Entity as the superclass. Additionally, the relationships between these classes exhibit aggregation ("has-a"), where one class holds a reference to another. For example, GameService references Game objects, Game references Team objects, and Team references Player objects. This structure demonstrates how instances of these classes are interconnected through aggregation.

## [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 application features a main driver class to initiate the creation of games, teams, and players, with the GameService class handling the process. Using the singleton design pattern, GameService ensures only one instance exists by restricting constructor access and providing a getInstance() method. Once active, the driver class can call addGame(), addTeam(), and addPlayer() methods, each using the iterator pattern to prevent duplicates before adding objects to their respective lists (games, teams, and players).

Game, Team, and Player are subclasses of Entity, which includes protected id and name attributes. The Entity class prevents null objects by using protected constructors, allowing only overloaded ones. The UML design highlights object-oriented techniques like polymorphism and inheritance through the Entity extension, as well as encapsulation and abstraction. For example, users can add teams via addTeam() without direct access to the Team constructor, maintaining simplicity and control over object creation.

**"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 easy accessibility and seamless server configurability. It features a user-friendly graphical interface and supports flexible terminal commands for enhanced functionality. | The platform is cost-friendly but can be difficult to navigate. It offers a command shell for simple server configuration and easy accessibility. | The server-side setup can be expensive, but the platform offers a user-friendly GUI and includes a command prompt for more advanced configurations and tasks. | Other devices often offer better specifications compared to mobile devices, whose specifications can vary widely depending on the user and device model. |
| **Client Side** | Mac systems can be expensive for users and require moderate time and expertise to set up and use effectively. Accurate skills are necessary to navigate the operating system efficiently. | Linux requires significant expertise and time to use effectively. It also necessitates knowledge of Linux data and commands, and can incur maximum costs for users, particularly for specialized configurations and support. | Windows systems are more expensive than Linux but are easy to learn and understand, making it simpler to support a Windows setup. They require minimal expertise to get started. | This platform provides flexibility for clients and developers to view updates from any location, though it can be slightly more difficult to implement compared to other devices. |
| **Development Tools** | Languages such as HTML, CSS, and JavaScript are fundamental for frontend development, supported by various libraries. Other essential development tools include PyCharm, GitHub, Visual Studio, and more, to enhance the development process and collaboration. | Languages such as HTML, CSS, and JavaScript are essential for frontend development, supported by various libraries. Additionally, Linux systems commonly use languages like JavaScript, Ruby, PHP, and Python for a wide range of development tasks. | Languages that  consist of HTML,  CSS and  JavaScript.  Libraries to  support frontend  and languages.  Developer tools  include Eclipse,  command  prompt, PyCharm,  Eclipse, etc. | Languages like HTML, CSS, and JavaScript are fundamental for frontend development, supported by various libraries. Integrated Development Environments (IDEs) for programming languages such as HTML, PHP, C++, and Python enhance the development process with tools for writing, debugging, and testing code. |

## 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**: To expand *Draw It or Lose It* to different computing environments, a scalable, secure, and cost-effective server platform is needed. The best choice is Linux (Ubuntu Server or CentOS) or a cloud-based service like AWS, Azure, or Google Cloud Platform (GCP). Cloud platforms allow the game to handle more players easily, while Linux is reliable, free to use, and offers strong security features. Windows can still be used for development, but Linux or cloud services are better for running the game online.
2. **Operating Systems Architectures**: The recommended architecture for the operating platform should be 64-bit, as it allows for better resource utilization and performance. Linux supports efficient multi-threading, making it suitable for handling concurrent user requests in multiplayer gaming. Adopting a microservices architecture would enhance performance and maintainability by enabling different components, such as game logic, authentication, and chat functionality, to operate independently.
3. **Storage Management**: For effective storage management, a combination of SQL and NoSQL databases is ideal. PostgreSQL and MySQL would be suitable for structured data such as user profiles, game scores, and leaderboards, while MongoDB or Redis would allow for fast, scalable storage of game states and chat messages. Cloud storage can be utilized for storing images, game assets, and backups. Linux’s file system options provide high performance and reliability for managing game data efficiently.
4. **Memory Management**: Efficient memory management is crucial for maintaining smooth gameplay and reducing latency. Linux-based systems use virtual memory and paging to dynamically allocate memory, preventing crashes due to limited physical RAM. Caching mechanisms such as Redis or Memcached can store frequently accessed data, improving response times.
5. **Distributed Systems and Networks**: Since Draw It or Lose It requires communication across multiple platforms, a distributed architecture is essential. A client-server model, where a dedicated game server allows multiple clients to connect via APIs, ensures smooth interactions. Load balancing distributes traffic evenly, while content delivery networks ensure efficient delivery of game assets worldwide.
6. **Security**: Security is a top priority, and several measures must be in place to protect user data. Data encryption ensure that stored user credentials and personal data remain secure, while TLS/SSL encryption safeguards communications between clients and servers. Authentication and authorization mechanisms provide secure login and session management, and role-based access control prevents unauthorized modifications.

Final Recommendation

For optimal performance, security, and scalability, The Gaming Room should use Linux or a cloud-based platform for deployment. A hybrid storage solution combining SQL, NoSQL, and cloud storage enables fast and reliable data management. Robust security measures, including encryption, authentication, and DDoS protection, will ensure data remains protected across all platforms. This approach guarantees that Draw It or Lose It can expand seamlessly while maintaining high availability, low latency, and strong security.