

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  2.0 | 7/21/2024 | Joshua Shults | Added functionality to add teams, players, and games to separate instances with iterator patterns to detect variables of the same name.  Changed recommendations based on requirements. Now recommending Linux and cloud services for server hosting. |

**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)

Draw It or Lose It is an interactive game application inspired by Win, Lose or Draw. In this game the teams will compete to guess what is being drawn from a library of stock images from the internet. The game will have four rounds per match, each round 1 minute long.

The application will ensure unique game and team names, manage multiple teams and players, and utilize a singleton pattern to make sure only one instance of the game exists in the program’s memory at a time.

The game will be web based and must be able to be accessed by multiple platforms. The server needs to be scalable to the amount of players engaged with the game.

## Requirements

* Game supports one or more teams
* Each team can have multiple players assigned to it
* Game and team names must be unique to prevent duplicate instances
* Only one instance of the game should exist in memory
* Unique identifiers for each game, team, and player on the teams
* Must be available on each platform
* Browser based

## [Design Constraints](#_2et92p0)

* The application must handle multiple teams efficiently, and needs to be able to support a large number of users.
* The system must support concurrent access by multiple users, requiring session management and consistency in data.
* Player data and game state must be protected against any form of breach (i.e. a same name instance running)
* Needs to be easy to use for players
* Is browser based

## [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 entity is the super class that each subclass inherits from. Each subclass also extends the Entity class and provides constructors for variables that will be used to identify them. To explain each class:

* **GameService: Manages game creation and iterating through arrays to ensure unique identifiers using the singleton pattern.**
* **Game: Represents a game instance, contains teams.**
* **Team: Represents a team in an instance, contains player list**
* **Player: Represents an individual player within a team**

**"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 a user friendly environment with good performance, but is at a higher cost and can receive less community support. | Linux is open source and highly customizable, but is not as widely used. | Windows offers broader compatibility for software, but is less customizable. | Mobile Devices provide convenience and accessibility, but has limited processing power and storage. |
| **Client Side** | Mac has a good development environment but comes with higher cost for their hardware. Also requires special knowledge for its OS as it is not as widely used. | Linux requires expertise in open-source technology. Development can be made easier through community and tool support, but is not widely used. | Windows is a common development environment, but development costs can be high due to licensing. | Mobile development requires cross-platform tools to be compatible with different OS, but mobile devices sell well in terms of apps like this one |
| **Development Tools** | Swift, Xcode | Java, Eclipse, Python, and multiple other open source IDEs. | C#, .NET, Visual Studio | Flutter, Android Studio, Xcode |

## 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**:
   1. The architecture of Linux (Ubuntu Server) is well-suited for server-side operations. It has a monolithic kernel which handles the entire operating system processes, like process management, file system management, memory, and device drivers in a single address space, which means fast load times and strong performance and stability. Linux is also open-source and highly customizable which itself provides excellent scalability to meet potential high server traffic for the game. Linux is also quite cost effective, it provides an OS with no license fee but with optional low-cost support packages which would lower costs concerning IT personnel. Linux is already widespread in cloud environments, which I will be recommending for server hosting due to its scalability and low start-up costs, as well as “pay as you go” models to reduce or expand depending on Draw It or Lose It’s player count.
   2. As for the development side of the game, I recommend using Windows. The Windows environment provides broad cross-platform capability, which is essential for a web-based game like Draw It or Lose It which is intended to be distributed across multiple OS platforms.
2. **Operating Systems Architectures**: The architecture of Linux is great for the server-side operations of this game. Its monolithic kernel will allow for efficient handling of game instance processes, fast memory management, and handling system calls which are crucial for a well performing game server. Linux’s architecture is designed for stability and has many security features which will be discussed in number 6. As for Windows on the game development side, it offers several robust and cost-effective development tools like Unity and Visual Studio. Unity also is compatible with OpenGL which is perfect to build the game in as it is a widely supported graphics API that works on most platforms such as Windows, macOS, Linux, as well as mobile platforms (iOS and Android.) OpenGL does not provide the best performance, but since this game will only be loading 2D photos in an instance of the game this will not be a problem.
3. **Storage Management**: For storage management I recommend Google Cloud for its “pay as you go” plans which ensure scalability as well as keep costs specific to the volume of traffic Draw It or Lose it sees during its release. Google Cloud also has its own SQL Server which can be used to efficiently store and retrieve large volumes of data the server will contain for user names, game states, etc. Google Cloud also offers automated backups along with its seamless scaling, and it is highly available. As for the internal server, Linux’s virtual memory management efficiently handles large datasets by using paging and swapping which will aid the server maintaining performance and stability during peak usage intervals.
4. **Memory Management**: Linux uses advanced memory management techniques such as paging, segmentation and virtual memory, which makes it suited to manage the memory requirements of Draw It or Lose It. With paging the operating system divides the physical memory into smaller blocks called frames where its virtual address space is divided into pages. When a process is executed pages are loaded into any available frames in physical memory that has access to addresses stored in page tables, making for easy swapping which reduces load times which is great for a game server. Segmentation works similarly, but is more about the logical structure of a program and instead of fix-sized units segmentation divides into variable-sized segments based on a program’s needs. Linux’s virtual memory also isolates each process into its own virtual memory space which enhances security. It also only loads necessary parts of a process into physical memory which reduces the overall memory footprint.
5. **Distributed Systems and Networks**: To enable Draw It or Lose It to communicate across the various OS platforms, a distributed systems approach utilizing Google Cloud’s load balancing is what I recommend. This service distributes server traffic across multiple instances, which keeps stability and lower load times even during peak traffic. For real-time communication between the client and the server, WebSockets are ideal due to their ability to maintain low-latency connections which will be essential to Draw It or Lose It so a guess at the last second will still be communicated to the server. Additionally, the nature of Google Cloud’s setup provides redundancy and fault tolerance, which minimizes server downtimes and keeps the game running consistently.
6. **Security**: Security will be important to consider to protect user information as well as not allow an instance to be modified. I recommend using encryption techniques on all communications between the client and servers using HTTPS to ensure that data is kept confident and maintains integrity. When it comes to authentication and authorization, implementing Auth0 with JSON Web Tokens will provide secure and scalable user management across each platform. Google Cloud also comes with a configurable firewall to restrict access to only necessary ports for users, which further ensures safeguarding the server from unauthorized access. Linux also has systems like SELinux and AppArmor which protects sensitive data. The game itself needs to be coded with security in mind as well, separating admins from game users, this will ensure that anyone playing the game will only have access to functions related to playing the game rather than retrieving data that is not associated with their username or game state.

**Development**

Unity is a cost effective game engine that can be used to implement 2D Assets into a customizable UI for players to interact with. The game can be built for WebGL to be cross-platform for each of the desired operating systems. For debugging and easier script editing, Visual Studio can be configured as the editor in Unity. For the team size needed for this project, Visual Studio has cheaper versions for less cost. Unity also has a “Play Mode” which can be used for in-house and external testing of the game’s features, as well as server testing.

By following this approach Draw It or Lose It can be developed efficiently and hosted on multiple platforms.

**Estimated Costs**

* Unity Pro is $1,800 per year per developer, which is a low cost for a game that will not require lengthy development time or need as many developers.
* Windows 10/11 Pro per developer is a one time fee of $199
* Each development machine would be roughly $1,000 including the monitor.
* Two teams would be necessary, one for the server and another for the game itself. Considering the requirements of the company, neither would be a large dev team.
* Visual Studio Community is free and suitable for this game.
* Google Cloud Hosting is approximately $1,000 per year per instance, including the server, storage, as well as their cloud SQL database. Cloud services is recommended due to lower start-up cost as well as cheaper maintenance, as an IT will not be required. Cloud is also scalable with the volume of users as Google Cloud offers a “pay as you go” model.
* Linux Ubuntu has no license fee for creating the server that the client would communicate with.