

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  1.1  1.2 | 11/12/23  11/26/23  12/10/23 | Michael Broadstreet | Initial draft  Added Evaluation  Final 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)

Client wishes to publish an online team-based multiplayer game based on guessing images within a time limit as they are rendered. Client lacks local expertise required to set up the environment for a web-based game, and has contracted us for support.

## Requirements

* Games to include one or more teams. Upper limit?
* Teams are comprised of multiple players. Upper limit?
* Game and team names must be unique
* No more than one instance of game to run at any given time.

Question for clarification: Does client wish for only one instance of the game server to be running at once, thus allowing for multiple concurrent games? Or do they only want one game to be played at a time. They would be capping their potential engagement in this case.

## [Design Constraints](#_2et92p0)

* Unique identifiers for players, teams, and games to prevent duplication.
* Use of design patterns to ensure single instantiation of game; singleton pattern an option.
* Language: Java, for maximum cross-platform compatability. Benefit: only one team required.
* For a time-based contest with potential players across the world, latency could be a concern.
* Security: there could be a possibility of client-side intrusion to prematurely decipher the image. Images should include some form of encryption or separation from client code.

## [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)

This design strongly leverages inheritance and polymorphism. Games, Teams, and Players all inherit from the Entity object, allowing them to use common methods for construction, and getters for name and ID.

The GameService object utilizes a singleton pattern to prevent multiple instantiation. This is an example of abstraction (hiding internal mechanisms) and encapsulation, as this object groups and hides much of the data that is used in the operation of the game, allowing the assignment of unique identifiers to the other Entities.

Finally, ProgramDriver object holds our ‘main’ method. This is where the GameService is instantiated, and the rest of the program is guided. It’s also where we can call the SingletonTester object, to test ensure our objects are behaving correctly.

**"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** | As of this year, dedicated apple server software is being phased out, though some functionality remains in the MacOS.  Hardware tends to run more expensive, with desktops often in the $2000+ range. Apple hardware tends to be more geared towards multimedia performance, potentially making it overkill for a server, and also less scalable than a Linux setup. | Open source, meaning the operating system is free, including many assorted applications and utilities. Constantly evolving ecosystem.  Known for stability, reliability, scalability, and security.  Utilizes a strong user privilege/permission model, and vulnerabilities tend to be quickly identified and patched by the community. | User-friendly interface, and Compatibility with the vast windows suite of applications and utilities. Easy to set up.  Not open source, so licensing costs will be higher than linux. Cost: $99-199 for a home pc license, $500 to $6000+ for windows server software.  Similar hardware to a linux system, but potentially less efficient use of resources. | Not a common platform for hosting a server. Could be downtime issues based on connectivity. Could potentially leverage AWS or Google Cloud services for scaling.  Fewer available server-side tools, due to it being a less popular option. |
| **Client Side** | Must align UI with the OS look and feel, per macOS design guidelines.  Primary languages: Swift, Objective C  Roughly 21 percent of desktop market. | Wide range of community supported and open source development tools and languages.  Potentially wide array of user desktop environments, so design consistency will be a consideration.  Roughly 3 percent of desktop market. | Default choice, low barrier of entry.  Languages: C# and .net  Visual Studio can be licensed for $499 – $6000+ depending on features.  68 percent of desktop market. | Consider cross-platform compatability, or choose iOS vs Android ecosystems.  Performance: mobile devices are generally more limited than desktop systems re: processing power/graphical capabilities.  UI must be designed with touch interactions in mind.  Massive Market:  300 million+ smartphone users, in US, roughly 60/40 iOS to Android. |
| **Development Tools** | Primary Languages: Swift, Ojective-C. Also: C/C++, Python, Java  IDE’s: Many, including Xcode, Visual Studio Code, Eclipse, PyCharm. | Primary Languages: C/C++, Python, Java, Bash  IDE’s: VSCode, Eclipse, Atom, PyCharm, and many more. | Primary Languages: C#, C++, Java, Python, Visual Basic .NET, PowerShell.  IDE’s: Visual Studio, VSCode, Eclipse. | For iOS, Xcode allows for building and testing of mobile apps on a Mac. $99 per year.  For Android development, Android studio is the official toolset  Cross-platform options: Xamarin, via Visual Studio can deploy to both iOS and Android. $25 -$1899 per year depending on scale. |

## 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**: We recommend building your Draw it or Lose it server on a linux platform. The money you will save on licensing and other fees can be better used to create a reliable and robust server. Additionally, linux servers are known for their reliability and security, key concerns when creating a distributed app.
2. **Operating Systems Architectures**: The linux kernel is the core of the operating system, which manages resources and interacts directly with system hardware. The hardware is abstracted through various libraries, to allow applications access without worrying about kernel-level programming. The linux Shell is a command-line interface that allows the user to input commands to be executed via the kernal.

While the backend server runs on linux, the front-end client app can be written in whatever languages Game Room wants to expand to, with the possibility of increasing their audience down the road. These client apps will cache and render the game’s images as required, utilizing the users own hardware.

1. **Storage Management**: For storage and delivery of the actual images, we recommend utilizing a CDN (content delivery network) such as cloudflare. This allows us to decouple the image hosting and delivery from our backend, preventing a potential bottleneck, while also improving responsiveness, reliability, and scalability.
2. **Memory Management**: Since images will be downloaded, cached, and rendered in the front-end app, this will free up memory as well as storage on the backend. The server will need sufficient memory to keep track of active users, active games, round timers, and scores. This will scale with number of users, so the modular and scalable nature of a linux setup will be an asset here.
3. **Distributed Systems and Networks**: The server and clients will communicate through the RESTful API. The OS-agnostic nature of RESTful allows for clients on any number of platforms, to include MacOS, IOS, Windows, Android, or Linux.
4. **Security**: To protect user information, the first step would be limiting what is collected to what is absolutely necessary. For the information that is collected, it should be encrypted and stored locally in a database on the server. The server should be appropriately firewalled, with regular system audits performed and IDS/IPS systems utilized.

We will also utilize a tiered approach to user accounts, utilizing role-based authorization. Users will only have access to functionality that is required to play the game, with higher roles reserved for system Admins/employees.

API keys should also be fully hashed/encrypted to prevent them from being attacked in transit, with SSL/TLS certificates purchased from Digicert.