

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 | 09/21/2025 | Josh Hall | Version 1.0 (updated 10/4/25) |

**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)

CTS has been contracted by The Gaming Room to help develop a web-based version of their game, Draw it or Lose it. Currently, the game is only available on Android. A game will have one or more teams, with each team having multiple players, and each game and team needs to have unique names. The game should only have one instance.

The singleton creation pattern has been implemented for object creation. Using the singleton will prevent the creation of multiple game instances. To address the need unique name, an iterator pattern has been adopted to prevent duplicate game and team names.

## Requirements

* The game must be available on a web-based platform
* The games and rounds should have specific time limits of 60 seconds per round, with a 15-second window per team to guess if the active team fails to solve.
* During the round, the drawing should start at the beginning of the round and be complete at the 30-second mark.
* A game will have the ability to have one or more teams involved.
* Each team will have multiple players assigned to it.
* Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
* Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.

## [Design Constraints](#_2et92p0)

* The game is being developed for a web-based environment, which will cause some constraints in the form of networking, compatibility, and security.
* Allowing players on Android and web-based devices to play together will cause additional compatibility constraints; selecting Java will help mitigate some of these issues
* Only using one instance of a game, taking care to be sure that this is implemented properly
* Unique identifiers, taking care to be sure that this is implemented properly.

## [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 diagram below shows the domain model. The ProgramDriver class will be the entrance point for the program; this is the standard procedure for Java. The program will also consist of a GameService class, Entity abstract class, Game class, Team class, Player class, and a SingletonTester. The Entity class is an abstract class that will be used as a parent class for the Game, Team, and Player classes as it contains properties and methods used by each of the three children classes. The GameService class uses a singleton pattern to ensure that only one instance of a game can exist at one time. Any given game will be an instance of the Game class. Once the game is created then the Game class creates instances of of the team class to facilitate the teams in the game. Once the teams are created the Team class will then create instances of the Player class for each player participating on each team.

The UML shows that each principal of OOP is at work in this program. The Entity class and it’s children classes make use of inheritance and Polymorphism. Abstraction and encapsulation are both used in the addTeam and add player methods, as instance creation is protected and done through methods.

**"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** | Apple no longer offers a server product | Linux is an open-source OS, which has led to a wide range of offerings and can vary greatly in price and feature set. Given the existing Java code, Linux would be a smart choice.  Ubuntu would be a good free option with optional pro support costing $500.00 per year. For something more enterprise-focused, something like Red Hat can cost north of a thousand dollars. | Windows Server is not ideal for this project as it is geared towards Microsoft-based tech stacks; in this case, that would mean moving to C# and the ASP.NET framework.  In addition to the less-than-stellar compatibility, Windows server can be quite expensive, ranging from about $1000.00 to over $6000.00 | Mobile devices should not be used for this type of application |
| **Client Side** | Developing for Mac OS typically means investing in Apple computers running Xcode. It also typically means Objective-C and or Swift. These languages are less common and may result in development bottlenecks. | Linux really has no limiters for language and will work well with the existing Java code. The downside of developing for Linux is that the install base is extremely small and uses many different distributions, which may cause some issues for development and may not be worth developing for. | Windows is the most worthwhile non-mobile platform to develop for due to its large install base. Nearly any compiled language can be used to develop for Windows, but ASP.NET is Microsoft’s favored language as it was created by Microsoft | Mobile devices are the largest user base by far, but are divided between Android (Google) and IOS (Apple). For Android, the SDK is based on the JVM. IOS is developed in Swift. |
| **Development Tools** | The most common IDE for Mac is Xcode, while Xcode is free to download a $99.00 fee is required annually per developer to get the full feature set. | Linux development has many options. If Java is desired, then Eclipse would be a good choice, with the added benefit that it can be used to develop for languages like C, C++, Python, and more.  If a more restrained or customizable development experience Then something like VI, Vim, Neo Vim, or Emacs can be used for basically any language | Windows has many different free development tools available. Tools like Eclipse, or IntelliJ, but the segment is dominated by Microsoft's VS Code. Microsoft also offers Visual Studio, but it is not nearly as popular | As stated above, Android is based around the JVM. The preferred development language for Android is Kotlin, but Java is a valid option. Google also offers Android Studio as a free IDE, though other IDEs support Android development.  IOS is programmed in Swift and Objective-C. For the best experience, the dev team will need to be on Macs and will need to be using Xcode which is listed as $99.00 per developer |

## 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**: Given the wide range of devices and OS I would recommend using a Linux based server. Linux has numerous cheap or even free server options that all have great documentation. APIs would be used to connect to the frontend. Linux is the most popular server OS in the world this has allowed for a strong ecosystem of tools and software to build up around it. Deploying on a cloud infrastructure would enhance scalability, I would recommend AWS or GCP.
2. **Operating Systems Architectures**: The recommended architecture would be to have the game state managed by the server and rendering would be done on the client side, this will allow for the server to maintain an authoritative state while minimizing data usage and help reduce cheating. Making use of something like Docker could further increase the scalability.

Client side rendering makes a lot of sense from a cost standpoint but it is also a good choice for performance. By having the client device handle the rendering, and by handing down a list of what images will be used for the round, Network related issues can be minimized. This approach would be viable both as a web based game or in app form.

1. **Storage Management**: If a cloud service provider is not being used I would recommend that SSDs, preferably NVME for best performance, be used in the servers for operations that require speed but for mass storage I would recommend HDD for the low cost. If a cloud provider is being used then this is a non-issue.
2. **Memory Management**: Linux has many built in features that help manage memory such MMU and page tables; MMU maps virtual memory to physical memory using page tables allow game processes to be isolated and to save resources by reclaiming resources from unused pages. This works well with demand paging which uses a similar algorithm to the least recently used algorithm to determine which unused pages should be ready in main memory. Linux also make use of transparent Huge Pages which reduce the pressure caused by large data caches, this primarily affects the Translation lookaside buffer which stores recently translated virtual memory addresses, by reducing the amount of entries to the buffer. Numa awareness to help group related processes together to reduce cross node latency.
3. **Distributed Systems and Networks**: To achieve a distributed set up I would use geo-aware load balancers with CDN to route users to a nearby endpoint. I would also continue to have the backend manage the game state and a stateless front end server performs lightweight checks and some validation events and tracks the match cycle. Ideally this would be a horizontally scalable environment that can respond to the work load dynamically. By using regional nodes it should be possible to have games fail over to the next nearest node in the case of an outage, It would be prefered if a second node in the same region would be the ideal. To minimize these types of issues and others health check and automatic mitigation are need to catch these issue as soon as possible. For connectivity I would recommend either using UDP/QUIC or TLS with WebSockets if UDP/QUIC for two way communication. Microservices would be used to connect everything together through APIs.
4. **Security**: For server security we will use the principal of least privilege for server access. This will mean that a role based hierarchy will be needed and the subsequent functionality required to administrate it. Data will be encrypted wherever possible, using something like AES-GCM. Modern TLS version will be required and version older than 1.2 will not be allowed. Using short lived session tokens, edge routers, private subnets all help reduce network attack surfaces. When the data is at rest full disk encryption with a KMS and strict rotation policies. All PII should be encrypted to ensure that plain text sensitive information is not available. Fully encrypted backups should be kept isolated from the main system and will require strict access control will be in place. To authenticate user accounts MFA will be used as will things like OAuth2 and OpenID. Features like secure boot, SSH, namespaces, and automated testing for updates will help protect the server from attacks. The overall goal is to minimize an attack surfaces as much as possible