

<Name-of-Software-Application>

# **CS 230 Project Software Design Template**

Version 1.0

## 

## 

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_gjdgxs)

[**Table of Contents 2**](#_30j0zll)

[**Document Revision History 2**](#_3znysh7)

[**Executive Summary 3**](#_2et92p0)

[**Requirements 3**](#_tyjcwt)

[**Design Constraints 3**](#_1t3h5sf)

[**System Architecture View 3**](#_4d34og8)

[**Domain Model 3**](#_2s8eyo1)

[**Evaluation 4**](#_17dp8vu)

[**Recommendations 5**](#_26in1rg)

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## [Document Revision History](#_lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 2.0 | 2/23/2025 | Austin Richmond | Made changes to the evaluation table. Also made changes to Recommendations. |

## 

## [Executive Summary](#_35nkun2)

**Problem**:  
The client, The Gaming Room, needs to develop a web-based version of the game **Draw It or Lose It (Currently on Android)**, where multiple teams with unique names and players interact over several rounds of timed play. The application must handle:

* **Multiple teams** and **players**, each identified uniquely.
* **Game and team name conflicts,** meaning they must be unique.
* **Accurate game progression**, including a fixed number of rounds with correct timing.
* The game **must only have one active instance** at any time and must track individual progress within that instance.

**Proposed Solution**:  
To solve these problems, we will develop an object-oriented program in which:

* **Entities** like **games**, **teams**, and **players** will be represented as objects with unique identifiers.
* A **singleton pattern** will ensure that only one instance of a game exists at a time.
* Unique names for games, teams, and players will be enforced by the system.
* **Game flow**, including round timing, will be tracked via internal logic in the game application.

## 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](#_1ksv4uv)

First would be the scalability. There will be many more users compared to the previous iteration because it's currently only on Android, and will now be accessible for all devices. Doing this you will need to ensure that the game is optimized in adding teams/games for faster/more efficient data querying.  
  
With this addition, the real time interactions needed for the game could cause the browser to lag behind when many games are being played at the same time. Real time game state updates are needed to be updated instantly in real time so something with a low-latency communication should help.

Just with the android version, this version will require even more data retention for when the game crashes to help players get back in. Will need persistent database that can withstand server failures.

## 

## [System Architecture View](#_44sinio)

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](#_2jxsxqh)

Game, Team, and Player inherit from Entity. This means they also share common attributes, such as id and name, also methods such as getId and getName. Because of this being allowed we are able to get rid of redundant code we would have to type for each class. So having a parent class and subclass helps with readability and processing. Game aggregates Team because a Game can contain multiple Teams. Same for Team aggregating Player. Teams can have multiple players. Same can be said about GameService is able to have multiple games. GameService uses the Singleton design pattern to ensure that there is only one instance of this class throughout the application, responsible for managing all game data.

**"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](#_z337ya)

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** | The good thing about Macs come with the secure unix architecture but they are expensive proprietary hardware and limited user customization + control. | Linux is super customizable because of it being open sourced. Expect high performance and cheap solutions to problems but IT BEING SO customizable and open make it harder to use from the start and require more skilled workers. | Windows is a good middle ground between them all. Its well known and used and is easy to access with a few limitations. Most software works on this one so tools needed should also work on this. Stability and security to take a hit on this option. | Best thing about this option is the accessibility and convenience for users but the drawbacks are limited resources and performance constraints. Things require heavy server side processing are not ideal. |
| **Client Side** | Main things needed for mac, because of the hardware exclusivity, are specialized developers. Lower overall user base makes it more costly to find those people. | As before, open source allows overall cost to be low. Drawback is similar to mac where you need specialized people. Most people have not used it so extensive training could be needed. | Most games are made on windows already. Supports many platforms/peripherals etc. Development should be cheap with ongoing software updates. | Development here has to focus on optimization. You get the wide amount of people but also the wide amount of hardware. The fragmentation in device specs/screens/OS ETC, requires a TON of said optimization and testing to ensure optimal performance for all users. |
| **Development Tools** | Major ones I would use here are Unity or Unreal Engine. Able to be used cross platform. | Have more options. Python, Java, Unity, Unreal Engine. Python and Java are good because they are also multiplatform and work well with 2d games, like we are, but Unity/Unreal still best 2. | Similar to Linux but add in C#. Good for Unity. Still would pick Unity or Unreal Engine. | Few different options but standout ones would be Java or Unity. Both are usable on all platforms. Java being weaker. |

## 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**: From everything I have read and have experienced I would say Linux would be the best OS. It is really flexible and can be used for a lot of different computing environments. As most already know Linus is also respected to have one of the best security frameworks. It also has a lot of frequent updates with crazy stability, so with Draw it or Lose it needing high uptime, Linux makes it the most reliable choice. Lastly it's also known for being pretty open source so solutions to problems could already be made and pretty readily available. Example of a known platform is the Steam Deck, it runs on Linux and is in the handheld world, being able to easily go into that market that taps into the steam deck and also steam would be a good segway into other environments.
2. **Operating Systems Architectures**: I would say the foundation for Linux rides on its modular architecture. Examples of this structure would be Linux kernel/shells/libraries. Nothing too complex about the kernel, it just is responsible for interfacing with the hardware…but the efficiency from Linuxs is what makes it stand out. The shells are pretty similar to the command prompt (windows) but Linux shell allows for more “scripting capabilities”. This helps with maintenance and deployment automations.
3. **Storage Management**: For storage management I would recommend ZFS. It has been around for quite a while and has been reliable. Examples of what ZFS offers are, data integrity checks, snapshot capabilities (super important for games), and good data compression (excellent for the type of game Draw it or Lose it is). Overall this would be my go to for the large multimedia data we will need.
4. **Memory Management**: Linux memory management relies on its caching, virtual memory, and paging. These help with performance and stability when gaming. Examples of this are, say, virtual memory. It allows the game to transfer inactive data to disk storage, freeing up the active use of RAM for more important tasks. Caching stores, temporarily, frequently accessed data and resources in a faster storage so this would help with the load times between rounds AND hopefully reduce latency for a game that needs instant inputs.
5. **Distributed Systems and Networks**: Two important things to note for this would be things such as microservers + API will hopefully alleviate some issues with outages/latency challenges across platforms. Example is docker for Linux. Basically it just makes sure the applications dont interfere with each other and can run constantly across various environments. Now, for API, services and protocols like HTTPS and WebSockets facilitate secure and real-time communications. So overall having things more load balanced and less redundancy help with this issue.
6. **Security**: Linux, the recommended operating platform, provides robust built-in security features including mandatory access control. Main things here that are good are the secure authentication mechanisms it offers. Examples are QAuth and multi-factor authentications. Routine patch management and security updates are critical practices that help against cyber attacks/breaches across all platforms.