

The Gaming Room

# **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 | 11/11/23 | Matthew Berdecia | Changes have been made to the cover page, as well as the revision of the document. The design, constraints and the domain model of this document. |

**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 is a project that’s purpose is to develop a game that is web-based that can be accessed on multiple platforms. The current model is draw it or lose it, which at the time can only be accessed on Android. The purpose of this game is to have a number of teams, that consist of several people going for 4 rounds, a minute each.

## [Design Constraints](#_2et92p0)

1. Must be able to run on both platforms
2. Each team should have a certain amount of players.
3. As for instances, the game can only be happening once at a time.
4. Each team should have their own assigned name

## [Domain Model](#_8h2ehzxfam4o)

Entity will create a relationship between:

Game, Player, Team

This means they all inherit or will get their information from Entity. With the use of UML we can show the principle of inheritance by making Entity a superclass. If we take a look at their relationship, we can see that Team & Player have a “type”. Meanwhile, Game has a Team, and GameService has Games. In the use of UML, we label it as aggregation. This means that it is an instance of a certain class and is referred to an instance from a different class.

**"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 is easily accessible and has great server customization. Easy to use, user interface.  **Update:** When it comes to the backend, starting your own project may cost significantly less, by running a type of websocket provided by html that will provide a client-side API to connect the browser with a node.js or a backend server. | Cost friendly. However, more difficult to navigate the platform.  **Update**:  Linux server uses hardening practices that may include strong and unique passwords, multi-factor authentication that will keep the client’s information safe. | Built in command prompt, user side is expensive. User claims easier use when using web extensions and SQL support.  **Update:**  A windows server solution may also start at minimal costs rolling a personal Node.js or Flask server authentication through React-native web based front-end. Some improvements may be seen within ensuring minimal downtime with DDoS mitigation, spoofing protection, bad port scanning, and host-based firewalls. | Specifications are better in other devices. While there is also the option to code the backend of the mobile app via a host, while it wouldn’t work well with a specific skillset. |
| **Client Side** | May be expensive for some users. Certain skillset and experience required to navigate the OS.  **Update:**  We could consider hiring a react-native web developer for an affordable budget. A hybrid app can be created rather than a native app – allowing faster speed to market and cheaper origination costs | There is a lot of experience required. Linux data is required to use the operating system. | More expensive than Linux systems. Easy to learn and understand | Provide flexibility to clients or even developers |
| **Development Tools** | Languages that may consist of HTML, CSS and JavaScript. These application are consisted of a set of different built-in tools, that include PyCharm, Visual Studios  **Update:**  **Flask, node.js, Websockets, mySQL** | Languages that consist of HTML, CSS and JavaScript. Libraries support frontend of application.  **Update:**  **Firebase, Amazon AWS, Heroku** | Languages that consist of HTML, CSS and JavaScript. include Eclipse, command prompt. | IDE’s for programming languages consist of HTML, php, C++ and Python. |

## 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**: The most appropriate platform that will allow The Gaming room to grow and develop Draw It or Lose It to other computing environments is Windows. This is the best option with all the tools that are up to our use.

**Backend (server) Recommendation:** Mac x64-based server. Since the game is Android, Web, and IOS-based compatibile, compiling and testing code requires Apple’s system, and thus should be done from a Mac-based server, easing IOS development and allowing the use of Xcode for memory management.

1. Overall, there are an abundance of IDEs that can be used with Windows, and the total cost to utilize it is typically lower.
2. **Operating Systems Architectures**: Microsoft Windows is a graphical os developed and published by Microsoft. Allowing users to store files, play games, surf the internet & many more things.
3. **Storage Management**: Aside from the computer’s hard drive, Microsoft also provides a cloud where we are allowed to store information as much as we needed.

**Recommendation for image server:** Amazon AWS both have great storage features, which will make scaling, backup, and security easier and more cost-effective long-term. They have pricing that will allow for highly scalable, fast, and relatively inexpensive data storage.

1. **Memory Management**: The Windows 10 storage sense would allow for storage and management of Draw It or Lose It photos and game players. Windows has Memory Management built in as a system utility that helps users store more information.

**Recommended technologies and approach**: Utilize web development and mobile application caching techniques, caching images as in-memory objects, serving images via Amazon’s AWS will allow faster downloads by reducing throttling done by browsers on the server.

1. **Distributed Systems and Networks**: Network based multi-user interaction systems such as network games typically include their own database shared among the players that are physically distributed and interact with one another over the network. A strong server network would also be needed, as this game’s success depends on multiple clients connecting to a single server to play one game altogether.

**Function As a Service**: With a high number of players, we may need several replicated servers, which sends connections to a Heroku-based Function As a Service to help load balance users across regions for matchmaking in game-rooms. There will be a database server to store user credentials, images can be stored and an event-based Flask server for the Drawing and chatroom logic.

1. **Security**: Windows comes with built-in security protection software. Although users can also use alternative source to protect the information.

**Function As a Service**: With a high number of players, we may need several replicated servers, that will allow users better connectivity. We will have a database server to store user credentials, images can be stored in an Amazon AWS-bucket, and an event-based Flask server for the Drawing and chatroom logic. Windows defender being one of the principal ways that is used to protect sensitive information within windows users.