

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_l6ti7uoag22u)1

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

[**Document Revision History**](#_grjogdjh5fi8)2

[**Executive Summary**](#_sbfa50wo7nsh)3

[**Design Constraints**](#_2et92p0)3

[**System Architecture View**](#_ilbxbyevv6b6)3

[**Domain Model**](#_8h2ehzxfam4o)3

[**Evaluation**](#_2o15spng8stw)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 9/15/2021 | Brandon Janisieski | Added Executive Summary, Design Constraints, and Domain Model |
| 1.1 | 10/5/2021 | Brandon Janisieski | Added Evaluation, and 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)

The Gaming Room wants to set up a web-based game which serves multiple platforms, Draw It or Lose It. The game should render images from a large library over a period of 30 seconds. If the guessing team does not correctly guess the image, the other teams have 15 seconds to give an answer. As per the client, the game should include the following features:

* A game will have the ability to have one or more teams involved
* Each team will have multiple players assigned to it
* The game and team names should be unique and allow users to check whether a name is in use already when choosing
* Only one instance of the game can exist in memory at any given time (create unique identifiers for each instance of a game, team, or player.

## [Design Constraints](#_2et92p0)

Distribution: How will the game be distributed? Obviously web-based, but will it be on a browser, will it be through a platform like Steam, Epic, etc.?

Maintenance: Will the game simply be maintained to continue functioning in the future, will it gain new features, etc.?

Capabilities: Does the program need to run on a some-what highly capable machine, or can it run on something with less power?

## [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)

In this UML diagram, “Game”, “Team”, and “Player” have an “is a” relationship with entity, meaning that the three of them are all entities, and all inherit from entity. The relationship between team and player is that Team has Players. It then follows that Game has Teams, and GameService has Games. GameService can have multiple games, and each Game can have multiple teams and each team can have multiple players.

**"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** | Offers some very nice features if using MacOS on the network, however, runs only on Apple manufactured hardware. Not a great option if already using Linux or Windows based software. | Linux is very stable and powerful and relatively easy to use. Linux is open source and has a lot of documentation and is free and can be run on effectively any hardware. | Windows, while not free, can be run on effectively any hardware like Linux. It also has a lot of documentation and a huge user base. Might not be as popular as Linux for server hosting, but still a very reasonable option. | Many option for mobile app hosting including AWS, Azure, Digital Ocean, and more. There are a plethora of choices for developers who want to host a mobile app, and each provides different features and capabilities based on price. |
| **Client Side** | MacOS is more expensive than Windows and can only (legally) run on Apple hardware. Much smaller userbase and not ideal for gaming. | Also not ideal for gaming due to low support, but can run on basically any piece of hardware with minimum requirements. Often used by hobbyists and is free. | The most used PC OS in the world, most people will be familiar with windows in some capacity, and it will run on effectively any hardware. Not free, but not as expensive as MacOS. | Android is the most popular mobile OS, taking up nearly ¾ of the global market. iOS is second making up most of the remaining ¼. Extremely large user bases in the mobile field, which might make it appealing if creating software that doesn’t have high system requirements. |
| **Development Tools** | Uses objective-c as the primary programming language. Good support for it, but not so much for other languages. | Linux has support for most programming languages, especially the most popular ones like C, C++, Java, Python, HTML, CSS, etc. Any issues are likely to be solvable as Linux has lots of documentation thanks to it’s open source nature. | C++ is the most commonly used language for games, however, Java, JavaScript, CSS, and HTML are also very popular for games and web based software. | Java is the most used programming language for app development on Android. For iOS Swiftic is one of the best frameworks available for iOS. |

## 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**: I recommend Windows as it has many of the benefits of Linux, with less of the downsides of Mac, all while having the largest user base (and therefore highest likelihood of familiarity for users) out of any of the major 3 PC OS’s. Windows has a lot of documentation and customization options, and therefore would be best for the purposes of The Gaming Room.
2. **Operating Systems Architectures**: x86 is the main operating architecture and is at the cutting-edge of what processors are capable of running on a computer. Powerful enough to support some of the most demanding software in the world, x86 is perhaps the strongest OS architecture available right now.
3. **Storage Management**: Storage can be done effectively two ways: Through SSDs for quick data transfer and load times, or HDDs for high capacity storage. SSDs have the advantage of being very quick in the file transfer capabilities and load times, where as HDDs typically have much higher storage capacity at a much lower cost. HDDS are not slow by any means, but a top of the line SSD will almost always be faster than a top of the line HDD. My recommendation is for SSDs, as typically file sizes will not be large enough for this kind of project to warrant the use of high capacity HDD storaghe.
4. **Memory Management**: As per Microsoft’s official webpage: “Each process on 32-bit Microsoft Windows has its own virtual address space that enables addressing up to 4 gigabytes of memory. Each process on 64-bit Windows has a virtual address space of 8 terabytes.”

Source: <https://docs.microsoft.com/en-us/windows/win32/memory/about-memory-management>

1. **Distributed Systems and Networks**: There are probably two different ways that The Gaming Room could implement their distributed systems and networks. Either through a client-server interaction, or a peer-to-peer interaction. A client-server interaction would mean that The Gaming Room would purchase space on a server (or maintain their own) for users to connect to and play the game. This means that users would interact with each-other through the server. Peer-to-peer would mean that one user (likely the one with the best/most stable network connectivity) would act as the host server while everyone else acts as the clients, connecting through the host. If the host were to leave suddenly, another player would be chosen as host until there were no users left.
2. **Security**: User information, especially private user information like emails, passwords, and credit card information, should be encrypted and protected from any potential malicious attacks. Users should have the option enable two-factor authentication in order to protect their accounts from targeted attacks as well.