

Draw it or Lose it

Joe Dempsey

# **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 | 11/13/2021 | Joe Dempsey | <Brief description of changes in this revision> |

**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 in the process of creating a game based off a game show where contestants guess the imagine before it fully comes into frame. They wish for multiple teams to compete against each other in the game and for it to be available on multiple platforms.

## [Design Constraints](#_2et92p0)

The design constraints for the Gaming Rooms Draw It or Lose It game are that game needs to have the ability to have multiple unique teams with unique team names. The ability to cross reference names on the server to see if anyone else is using that name or if that name is currently in play / playing the game. The system will also need to be able to have only one instance of the game on file.

## [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)

GameService, Games, Team and Player classes all inherit files from the Entity class. All of the classes will share common files and names between each other to prevent over complication of files within the java code. This way you won’t have to instances of the same thing asking for the same information. Example, the Player instance is shared between all classes so when you input information for Player all of the classes receive that information and are able to share and pull from it as needed.

**"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** | For Mac on the server side you must make sure the program is readable from Mac to the server SQL. Mac is upgradable like Windows and Linux but is not very user friendly. An additional requirement for Mac is to ensure that you have the necessary work crew that can handle server side difficulties with Mac based IOS. This can be a new release of IOS updates or a complete redesign of the application to be compatible with older and newer IOS. Being able to have employees who can handle multiple operating systems would be beneficial however it can be easier to have dedicated teams. One team each for Mac, Windows and mobile for example. Mac offers sever tools but you must make sure they meet the same requirements as the other server tools. It would not be very cost effective to be running a dedicated server for each operating system of your web domain or application. The best course would be to find a server client who can offer a server that can handle each different operating system and allow them to communicate with each other. This can also allow for cross platform play for the customer. | Most secured of the choices and the must popular in Server applications. Not very user friendly / requires more time to build the architecture of the system before the game files can be loaded on. Thankfully Linux is the most compatible of the operating systems and has dedicated applications for server deployment. Out of all of the operating systems Linux will be the easiest and most cost effective for deployment. | Most compatible from all choices. Comes with preloaded software for server applications, ie Windows Server. Not as secure as a Linux based OS but is more user friendly and comes with more features preloaded. Windows and Mac do not play well with each other but this can be mitigated by deploying a client that can read both and allow each to play with each other. This will be expensive but more beneficial in the long run. | Most popular for client use, will most likely be the way clients play the game. Not very good for hosting a server as the devices are mobile and not stationary. Harder to implement updates as Mobile devices have several different operating systems that require writing new code for every update. In addition mobile application will have the same challenges as Windows Mac and Linux. Like I stated in the Mac block having a dedicated team for each development side wouldn’t necessarily be the most cost effective but it will be the most error free way of going about it as you won’t have employees working on the same project in different types. |
| **Client Side** | Mac is the most expensive for a client to start using but is one of the most popular home computers for College age clients. Requires the programmers to be knowledgeable in translating coding languages to be readable on a Mac OS as well as other OS so that there is no differences between Operating systems for players. | Least popular of Operating system amongst clients but those that use a Linux based system are more likely to be knowledgeable with operating systems and will be more responsive is debugging issues in a beta or alpha test. Having Linux be the primary test platform for the project is the best as the people who tend to use Linux are more tech savvy and will be able to give more reliable feedback. | Requires the same expertise as Mac and Linux but has more features to help the developer create. Keeping Windows and Mac the same across the board is the best approach as it prevents unnecessary headaches from having different tools launched with different applications. | Most likely the most difficult of the operating systems for coding and end user. There are multiple operating systems that pertain to mobile user the most popular being Android and Apple OS. Cost will be the same as the other operating systems, but the time required will be more as there is more things needed to be created as apposed to a window based system. |
| **Development Tools** | Java, Python, C++, HTML, MySQL are all common languages that can be used for Mac. A dedicated Server for Mac or a server program that can interface with multiple operating systems and platforms. A dedicated team for development and update control. | Java, Python, C++, HTML, MySQL are all common languages that can be used for Linux as well as Linux based languages unique to that operating system. A dedicated Server for Linux or a server program that can interface with multiple operating systems and platforms. A dedicated team for development and update control. | Windows shares the same languages as Mac, Java, Python C++, Html, JavaScript, MySQL. A dedicated Server for Windows or a server program that can interface with multiple operating systems and platforms. A dedicated team for development and update control. | Mobile uses the same languages as Windows, Linux and Mac. A dedicated Server for Mobile apps or a server program that can interface with multiple operating systems and platforms. A dedicated team for development and update control. |

## 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 would recommend they start on a Linux based system as it is the most user friendly for developers and use Linux in development before moving to Windows for the end user applications and testing. From Windows the next would-be Mac and Mobile. However the best course of action for the game is to develop with the idea that it will be played on multiple different platforms. Starting on linux to build the base of the game then expanding to more operating systems is a good start.
2. **Operating Systems Architectures**: Windows and Linux have the most freed up architecture as their systems are completely customizable for the end user. Most out of the box windows computers come with on board graphics and processors that can handle most small scale less demanding games and programs with the ability to upgrade these components later on down the road to better ones.
3. **Storage Management**: I would recommend using a SQL based system for storage on either Linux or Windows. For storage recommendation SSD is the best choice hands down. Its widely aviable and far faster then traditional hard drives. Plus it opens up the possibility of using NVMe which is getting faster each generation. The only downside is the cost, high performance SSD can cost a pretty penny but what you get for that price is well worth it.
4. **Memory Management**: I would recommend writing to both a server and to the devices on memory to so that there are two copies of the game files. Using a windows based server is the best.
5. **Distributed Systems and Networks**: I would recommend publishing on Linux and Windows first as a web-based game that way it is easier to debug issues that come up in launch. Once the marketplace is large enough roll out mobile and Apple applications so that those users can enjoy the game. This way you can use the team between launches to make changes and develop more features for the game without having to be rushed.
6. **Security**: For security I would recommend any user information being encrypted to protect privacy as well as relying on in system encryption and end users on security software. You don’t want to go to hard on security as this could interfere with the end user’s security protocols on their system, but you want enough to protect the clients sensitive information. Encryption and backup files storage in inaccessible locations are the best route.