

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 | 3/19/2023 | Brian Engel | Initial 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 wants to develop a web-based game that serves multiple platforms based on their current game, Draw It or Lose It, which is currently available in an Android app only. We will have to make it so the program can run on multiple other platforms.

## Requirements

*Web-based game*

*Needs to run on multiple platforms*

## [Design Constraints](#_2et92p0)

Needs to be a web-based game

Needs to run on multiple platforms

Have to transfer files from android to other platforms

## [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)

Entity is the parent class of Game, Team, and Player. It extends the id and name variables to its children, as well as its constructors, getId(), getName(), and toString().

GameService is a singleton class. It contains a arraylist variable for games and a method to add (0 to many) games. It also contains variables and methods for nextGameId, nextPlayerId, and nextTeamId to give each instance a unique id number.

Game class has an arraylist variable for teams and a method to add (0 to many) teams. It also overrides toString() from Entity.

Team class has an arraylist variable for players and a method to add (0 to many) players. It also overrides toString() from Entity.

Player class inherits from Entity, and overrides toString().

The overrides of toString() are examples of polymorphism, the Entity class is an example abstraction, because you never actually have an instance of it, the getter and iterator pattern are examples of encapsulation, because it hides the internal parts of the classes, and inheritance is shown in the parent (Entity) and children (Game, Team, Player) classes.

**"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** | Stable.  Easy to use.  Good security features.  Automatically recognizes other apple products (if client is Mac or iOS)  Most expensive. | Linux is the cheapest of the 3 majors.  Stable.  Difficult to use.  Compatible with most webhosting software.  Not compatible with Windows applications.  Linux allows for running scripts written in PHP, Python, and other Unix-originated languages. | History of crashes and bugs.  Frequent rebooting.  Easiest to use.  High volume of activity can cause problems.  Middle of the road cost.  Windows allows for running ASP scripts and .NET and other Microsoft technologies. | Always have access to it.  Secure.  Cheap.  Cannot handle much traffic.  Cannot handle complex tasks.  Probably cannot handle large amounts of data.  Limited power supply. |
| **Client Side** | Secure.  Good support.  Easy to use GUI.  Can run Windows.  Default web browser is Safari.  Most expensive. | Hard to use since it is command line.  Stable.  Cheap.  Secure.  Not as much support. | Not as secure.  Good support.  Easy to use GUI.  Default web browser is Edge.  Moderate price.  Prone to crashes.  Most used operating system. | With small changes to the program to not have to use a mouse, mobile is ideal for client side.  Secure.  Cheap.  Always have access.  Cannot handle complex tasks (shouldn’t need to).  Probably cannot handle large amounts of data (shouldn’t need to).  Limited power supply. |
| **Development Tools** | Xcode (IDE)  Supports:  C  C++  Java  AppleScript  Python  Ruby  Swift | Eclipse (IDE)  PyCharm (IDE)  Visual Studio Code (IDE)  Supports:  C  C++  Java  Python  JavaScript  Swift | Has so many developer tools it would be impossible to list them all.  Eclipse (IDE)  PyCharm (IDE)  Visual Studio Code (IDE)  Countless others.  Supports:  C  C++  Java  Python  JavaScript  Swift (even though it is mainly used for mac) | Xcode (IDE iOS)  Android Studio (IDE android)  iOS – Objective-C or Swift  Android – Java or Kotlin |

## 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**: For the operating system I would have to choose Linux. Even though it is harder to use than Windows, it is more secure, and that is a major concern of The Gaming Room. Another advantage is that it is the cheapest. MacOS or mobile device servers would not be advised.
2. **Operating Systems Architectures**: Linux uses a kernel between the hardware, such as HDD, RAM, and CPU, and the shell, system library, applications, and system utilities. The kernel oversees memory management, process management, device drivers, and system calls and security.
3. **Storage Management**: I would recommend a NAS (network attached storage) such as Amahi. It is easy to setup, has tons of resources, and is open source and free. Once the game grows larger, we might consider upgrading to cloud-based storage management, since they can hold any amount of data, and usually charge based on how much an application uses.
4. **Memory Management**: Linux uses the implementation of demand paging and virtual memory***.*** It also has memory for internal kernel structures and user programs. Any processes called by the user will go through the RAM and into the heap. It is important to delete any objects created once they are not needed anymore in order to keep the application running efficiently.
5. **Distributed Systems and Networks**: Using a client-server model the application should be easy to implement. This is a distributed system because the server interacts with the storage management to access the images. The server is also doing the main processing for the program. There are multiple clients that ask for information from the server, and process the information in the web browser, no matter what operating system they are operating on.
6. **Security**: The first thing to do for security is to have usernames and passwords, and to have varying access to different objects for different types of users. This will allow for authentication and authorization to restrict unauthorized and under authorized users from being able to access important or delicate processes. Another security feature is the kernel in Linux because which is one of the reasons I chose Linux in the first place, the security is very good. Since it is open source, any vulnerabilities are usually discovered and eliminated very rapidly, unlike Windows or MacOS which have hidden source code. It also has a superior user privilege model. Another thing the application should do is “clean” the user input to make sure it is not malicious.