

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## 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 | 05/22/2023 | Brad Decker | Executive Summary |
| 1.1 | 06/9/2023 | Brad Decker | Added Evaluation |
| 1.2 | 06/23/2023 | Brad Decker | Finalized Recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

The software design problem is the environment for the game Draw it or Lose it. In specific, The Gaming Room needs an environment that can serve multiple games, teams and on different platforms. Because of the need to serve on multiple platforms I believe the best design solution is a Bridge Pattern.

## Requirements

* Game must be able to accommodate one or more teams.
* Each team will have multiple layers assigned to it.
* Game name must be unique.
* Team names must be unique.
* Players must be unique (name may be duplicated such as Steve on Red Team and Steve on Blue team).
* Only one game can exist at any given time.
* Because each game, team, and player must be unique a distinct identifier should be used for each instance.
* Ability to serve multiple different platforms such as Windows, Linux, and Apple devices (including the original serviced Android devices).

## [Design Constraints](#_2et92p0)

Since the player, team, and game can exist on different devices we can use a Bridge Pattern to bridge the gap between these devices. The bridge pattern should be able to satisfy the new devices that aren’t Android and create instances that exist on other devices as uniquely identified objects.

## [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)

The Entity class creates a relationship between the game, team, and player classes. The 0…1 references the relationship of inheritance from class to class. Each class has it’s own unique identifier noted as id and a name that flows back into the Entity class as the parent class to Game, Team, and Player.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Cost would be prohibitive. The pros are that it’s great for native Mac users. The biggest con though is the lack of scalability. MacOS Server lacks capabilities to adequately serve many clients at once which would be a major issue for our clients. | Ubuntu, a free server option for Linux users. This option has plenty of online support. The biggest problem with it is problems with software to hardware. It also may be hard to find someone to develop code for Linux. | The cost would be less than Mac but not free like Ubuntu/Linux. Windows servers are great for reliability and easy to find resources to work with them. Tools are lacking for these servers though and upgrading to newer versions can be time consuming. | Oracle Mobile Device servers are quite expensive. They are quite reliable and stable. However large data transfers are not that great on these servers. |
| **Client Side** | Quite a few clients use Mac computers and will find a lot of connections that are from Mac computers. Finding expertise in programming for Mac will be relatively easy. The time taken to create a Mac application should not be high and thus not costly. | Few clients use Linux, it’s used commonly for Web Servers. Finding expertise in programming for Linux will be relatively easy for the Web Servers but will likely be challenging for client-side programming. This may increase the cost to find someone to program in Linux. | Quite a few clients use Windows computers and will find a lot of connections that are from Windows computers. Finding expertise in programming for Windows will be relatively easy. The time to create a windows application should not be very high and thus not costly. | The number of variations in mobile users will be costly to maintain. The main considerations would be on Apple phones and Android based phones. The languages are various and converting between Android to Apple can be time consuming and challenging. However, in our day and age many clients likely have a mobile phone so the market may be larger and thus have value that outweighs the cost. |
| **Development Tools** | Languages for Mac include:   * C * Objective C * Swift * Assembly * AppleScript   Some IDE’s include Xcode and Visual Studio. | Languages for Linux include:   * C * C++ * Python   Linux has quite a few IDE’s including: Visual Studio, atom, Sublime, Eclipse | Languages for Windows include:   * C * C++ * Rust * Assembly * PowerShell   Commonly used IDE’s include Visual Studio, Eclipse, NetBeans, JetBrains | Languages for Mobile apps include:   * Java * Python * C++ * Kotlin * Rust   Some IDE’s include: Android Studio, Eclipse, Visual Studio Code |

## Recommendations

1. **Operating Platform**: Based on cost, security, and accessibility I recommend a LINUX server, to be more exact an Ubuntu Server. Ubuntu Servers are one of the most popular types of LINUX servers and they work for many different types of operations including web servers which is what Draw It or Lose It is going to be.
2. **Operating Systems Architectures**: The best operating system architecture for having multiple users access the same resources would be a Layered Architecture which LINUX is capable of supporting.
3. **Storage Management**: Ubuntu provides the Apt-Cacher-Server which can be used to do storage management of an Ubuntu Server. This can also be used in conjunction with distribution and can speed up some of the common elements needed for Draw It or Lose It.
4. **Memory Management**: Ubuntu splits into three zones for memory management, DMA, normal, and high memory. The DMA is just for direct access memory. The normal is used next and is hosted within the first 896 MB of physical memory. The high memory zone is anything more than that. Most of the allocations are done through slabs and are thus marked for what is currently used, what is empty and what is only partial. Some testing of the caching system is important to optimal implementation of the images used for Draw It or Lose It.
5. **Distributed Systems and Networks**: To complete network requirements we can use network-device drivers to support the interconnected devices within our system. We may also consider using a virtual machine to complete multiple different instances of the game on different platforms or operating systems. Using a virtual machine to do so would help reduce some redundancies of creating code for specific devices and provide flexibility to the clients on what they want to use to run Draw It or Lose It.
6. **Security**: The security for Ubuntu is similar to many other UNIX styles and can be grouped into authentication and access control. Authentication is used in many other platforms and applications already but Ubuntu allows for encryption of the root folders making it harder to access personal files without being that user. The roles used in most other security systems are similar to those in Ubuntu and are easy to set up for different users. Both of these together will make it easy to verify the users before they access files within the Draw It or Lose It game.