

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 | <09/10/23> | <Zachary Hancock> | Wrote out the executive summary, requirements, design constraints, and UML description. Implemented Entity class as well as additional features to its children for adding Teams and players to lists. |

**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 client, The Gaming Room, wants to expand their game, Draw It or Lose It, to more platforms and don’t know how to set up the environment for a web-based version. We must set up the web-based environment of their Draw It or Lose It game with the same functionality as the original.

## Requirements

* A game will have the ability to have one or more teams involved.
* Each team will have multiple players assigned to it.
* Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
* Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.
* Web-based

## [Design Constraints](#_2et92p0)

* The limitations would be drastically different based on the processing power of whatever device is accessing the game like mobile or PC/MAC
* Each platform will have different scaling based on their device mobile much smaller so it will have to not have a set size but instead be scaled per window sizing.
* How each platform handles running web-based applications on their client
* Tools will have to be acquired for each platform to be able to test and implement the web-based application on each device to ensure proper execution so that every user has the same experience.

## [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 a parent to Game, Team, and Player classes so these three classes will inherit its properties that they’ll be able to use if desired. Each of these children utilizes the encapsulation principle of overriding a method, in this instance, they each override the method “toString()”. The parent and children also utilize Encapsulation by making their variables private and implementing public accessor methods to allow something to read the value from the outside. The multiplicity relationship between GameService to game is that there may be 0 to many games for a GameService. There may be 0 to many Teams for each game and 0 too many players for each Team. Lastly, ProgramDirver has a usage dependency on SingletonTester to be fully operational.

**"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 paragraph response covering the indicated information.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | * More costly * More stable * User friendly UI * Must wait for developer to update and tweak. * More resource intense * Will need specific Mac hardware to run. * More secure due to limitations to the Apple ecosystem | * More Cost effective * Open source, more customizable, * More difficult UI * Stable * Less resource intense | * More prone to crashing than other OS * More costly for licenses * Easily understood UI. * Must wait for the developer to update and tweak. * More resource-intense | * Limited to less resources * Security is limited to developer updates. * Nonscalable hardware |
| **Client Side** | * More costly hardware * Higher performing hardware * Based on UNIX so can be easily compatible with Linux * Can be hard to learn UI if no prior experience, so can affect time if the developer doesn’t have expertise. * License fee * More stable | * Free * Open source so can be modified to suit needs. * Wide range of programming compatibilities * Huge programming community so no worry about expertise * Steeper learning curve | * Widely used to more expertise in this environment * Less costly and easy to scale up hardware. * Easy UI * License Fee * More downtime with restarts and errors can cause more time for development. | * Mobile being a newer space requires newer expertise to develop on such a platform. * Different hardware, different screens different limitations even on the same OS can pose time difficulties as adapting to different limitations can be difficult. * Ever changing landscape can put pressure on the budget as being on the forefront can soon become being drastically behind |
| **Development Tools**   * **Tools that apply to all of the categories would be apps like project planning tools that help make all of your work in one place for the team.** * **GitHub** * **Azure as well specifically in deploying web services** | * Access to Swift and object-c for Mac-specific coding, other than that has access to all other languages in IDE’s that offer it. * Due to programmers' popularity, it remains a diverse platform for development. * Microsoft Visual Studio, Xcode, PyCharm, NetBeans, etc. | * Due to its popularity just like MAC it also has a vast variety of IDEs to accomplish deployment to this platform. * Visual Studio, NetBeans, atom, etc. | * Like other MAC and Linux, it is very popular for programmers, so it has a large variety of IDE’s and tools. * Visual Studio, NetBeans, PyCharm, atom, etc. | * Being a newer platform, the more tailored IDEs are limited but still able to use the general IDEs that all platforms use like Visual Studio * Android Studio, Xcode, atom, Eclipse, NetBeans, * Flutter allows cross-platform coding without having to create code for each platform |

## 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**: Out of the considered operating platforms, I recommend Windows because of the following.
   1. It is the easiest scalable platform, windows can be very affordable to upgrade hardware that’ll be more futureproof, unlike MAC which doesn’t have any way to upgrade hardware currently.
   2. The most widely used, Windows has the widest range of support and expertise available if there is a roadblock within the operating platform present.
   3. Capability of cross-platform development, with the use of Android studio, emulators, or virtual machines to develop in another environment.
   4. Well-supported updates pushed with the Windows OS, being widely used the developers constantly push fixes and security updates to ensure a safe and efficient work environment.
2. **Operating Systems Architectures**: From the top down the architecture goes from apps and system services to the operating system kernel. The application level is the apps on the OS that the user interacts with which in turn interacts with the system services. The system services include things like input/output, VRAM, etc. that interact with other software components within the OS to accomplish its tasks. They operate without user interaction. Lastly, the operating system kernel is the base level which includes other low-level device drivers that are responsible for taking system requests from the system services levels and translating the requests into commands for the computer's hardware like the CPU and other devices.
3. **Storage Management**: As previously discussed, the best possible solution for storage for Draw it or Lose it will be a cloud solution. This will allow the game to scale up for more images easily, the game will have the ability to pull the images over the internet easily rather than pushing updates to the user every time there is a new image added. Specifically for Microsoft Windows Azure would be a good option as it's designed with Windows in mind, but it can easily support multiple platforms. It is set up as SaaS, PaaS, and IaaS so it can be scaled based on the use that is needed.
4. **Memory Management**: Windows uses VRAM where the processes are assigned a virtual address that is then translated to a physical one. They are translated to pages, which are blocks of addresses to not interfere with others. The kernel level processes get their own set of virtual addresses as well that the user shares a copy so calls to the operating system can be made from the upper level. Since users can't directly access memory allocation for protection of the system.
5. **Distributed Systems and Networks**: To communicate between various platforms there will have to be a centralized database to be the pathway for communication, as we practiced before a RESTful API would be suitable to work with varying systems. The server can be optimized for load balancing to allow for easy scaling of player counts, so it can offload unused resources an optimize needed ones easier. To ensure user information is saved a backup system will be in place to ensure constant saving of progress. With that in mind to enhance user connectivity during outages we’ll implement retry mechanisms to allow the server to handle the outage itself and try to smoothen it out before erroring out completely.
6. **Security**: To enforce user security, we’ll utilize a role-based access system to prevent intruders from easily gaining access to privileged information. In conjunction, we’ll use authentication and intruder detection systems to properly authenticate users and try to identify early threats to be dealt with.