

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## 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 | 09/19/2021 | Andrew Black | Initial revision |
| 1.1 | 10/03/2021 | Andrew Black | Completed Evaluation |
| 1.2 | 10/17/2021 | Andrew Black | Completed Recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room wishes to adapt their game, Draw It or Lose It, for the web. A game will have the ability to have one or two 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. Finally, only one instance of the game can exist in memory at any given time. This will be accomplished by giving unique identifiers for the game instance.

## [Design Constraints](#_2et92p0)

Developing for a web-based environment presents unique design constraints such as managing the size of the final served content, making sure the APIs are consistent, and that the application fits on the device requesting it.

## [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 main class is the ProgramDriver. It makes use of SingletonTester to test that there may only be a single instance of the game in memory at any given time. The GameService class manages adding and getting Game instances, which allows for the adding of Team instances, which allows for the adding of Player instances. Finally, the Game, Team and Player classes all inherit the Entity class to provide basic universal information such as the entity ID and name.

**"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** | You must worry about the overhead of macOS (user interface elements, processing and memory required for them) as well as any security features that might not be applicable to the application in question. For most (if not all) Mac machines, you cannot upgrade the processor without buying a whole new machine. Most Mac machines do not allow you to upgrade the memory as they are soldered in, but Mac Pros do. macOS is not free to install or use on any system that isn’t using Mac hardware. | The overhead of hosting on a Linux system is minimal. You do not have to worry about running a user interface (desktop environment) if it is not needed. The server administrator can change out hardware and allocate resources as needed because the operating system is heavily adaptable to all different types of hardware. It is an open operating system, meaning security can be audited by anyone at any time, and adjusted to the specific needs of the host. Linux is completely free, but there exist versions like RHEL that are non-free and enterprise oriented. | It is much like hosting on a Mac machine where you must worry about the overhead of the operating system and performance measures that might be taken. One added benefit hosting on Windows has compared to mobile devices and Mac machines is that the server administrator can change out hardware and allocate resources as needed. Windows Server is not free, but Windows itself can be used without a license. Restrictions may apply. | The most important aspect to consider is battery life, performance, memory, and connectivity. Mobile devices may not have the best wireless connection, they may be heavily limited on memory (as the system might allocate as needed), and the system may take necessary measures to preserve battery life. Mobile devices are not free but the operating system, Android, is free and open-source. |
| **Client Side** | Writing a native macOS UI application requires knowledge of the Cocoa language or Objective-C. There will always be a bar at the top of the screen to offer specific options to the user under drop-down menus. | Writing a native Linux UI application is difficult as distributions may use different toolkits, such as GTK, Qt, and much more. | Writing a native Windows UI application is more straight-forward as Windows provides the Win32 toolkit, Windows Forms, MFC, WTL, and more. | Writing a native Windows UI application can be more straight-forward than macOS, Linux, and Windows, as it can be done in Java through Android Studio. |
| **Development Tools** | Cocoa, Obective-C, and possibly Swift are the most common languages to write native macOS applications in. They can be done through Xcode, which is developed by Apple. Xcode is completely free, but can only be used on a Mac system, which is not free. | C++ is the most common language to write native Linux applications in, as it has been tried-and-tested for a long time with many different toolkits available. Any text editor can be used in addition with compilers such as GCC, Clang, or others. There exists IDEs to work with those languages such as CLion. | C++ is also the most common language to write native Windows applications in, but others have become more mainstream such as C# and Visual Basic. Microsoft provides Visual Studio to develop applications with multiple languages besides the ones listed above, such as Python, JavaScript, and more. It is completely free, but other editions (such as Professional and Enterprise) are offered at a cost. | Google provides Android Studio, a cross-platform IDE based on IntelliJ IDEA, to develop Android applications in. It is completely free. |

## 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**: Using Windows would be the best option given how quick and easy it is to start developing and virtualize Linux features (with WSL 2.0, for example), allowing for further cross-platform development.
2. **Operating Systems Architectures**: As explained above, Windows makes it quick and easy for developers to get started and continue development. It is a versatile yet dependable platform as its popularity continues to propel development and tooling for the platform.
3. **Storage Management**: Making use of general compression algorithms such as LZMA2 allows for a balance between performance and storage space. For images, it would be best to investigate the WebP container or traditional methods like JPEG for efficiency.
4. **Memory Management**: Windows allows for both physical and virtual memory allocation along with exploit prevention methods to aid in security. For developers, it would be good to note cases of manual memory allocation and points that may result in memory leaks.
5. **Distributed Systems and Networks**: With distributed systems, the possibility of congestion and outages are lowered as multiple networks can be set up as redundancy. One downside is that there may be discrepancies between the networks, like latency and caching issues, but the pros far outweigh the cons.
6. **Security**: Windows provides many mechanisms to aid in security like exploit protection regarding user account access and memory validation. Along with this, Windows provides systems administrators with adequate auditing tooling, such as Event Viewer and Group Policy. Developers should make use of trusted system tools to perform encryption and validation of data passed in and out of the network.