

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  1.1  1.2 | 09/16/24  09/30/24  10/14/24 | Joseph Roberts  Joseph Roberts  Joseph Roberts | Initial draft of summary, requirements and design.  Address any issues and review the evaluation section.  Address feedback and review the Recommendations section. |

**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 further develop their game *Draw It or Lose It* which is currently available on android. The new version would be web based and is loosely based on the 1980’s television game *Win, Lose or Draw*. The game will consist of four rounds at one minute each, there are multiple teams whose goal is to guess the puzzle picture before their time runs out, the opposing teams would also have a chance to guess the puzzle afterwards.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_2et92p0)

* Only one instance of the game can exist at a time.
* Game and Teams must have unique identities.
* Teams must have multiple players, and one or more teams will be involved.
* Possible cross-platform

## [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 creation of the *Entity* class acts as a parent class to the Game, Team and Player classes. This allows them to inherit information from the *Entity* class. The *ProgramDriver* class uses a *SingletonTester*, this is to test the code and keep only one instance of the code in memory. The lines in-between classes show numbers and its associate for example:  *Game* will have zero or more association with *Team*. The UML is a basic roadmap during development.

**"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** | Mac’s hardware and software tend to offer the highest security and reliability. Can run Windows in a VM for testing of projects with complete compatibility. | Linux is customizable and it is targeted less frequently for attacks allowing stronger security. | Windows allows for a vast range of compatibility, user-friendly, regular updates and business and enterprise support. | Portable, touchscreen, and hardware capabilities. Android devices are built upon Linux, as iPhones utilize Mac. |
| **Client Side** | Macs are expensive for users, limited hardware options, not ideal for gaming performance. | Being open source allows for lower cost upfront, but major version updates can cause complex issues, command line operating instead of GUI, requires knowledge and experience of OS not suitable for new users | Can be slightly costly, privacy concerns and can be resource intensive. One of the most common and familiar OSs’ allowing for customization options for a vast range of users and automated updates for those less familiar. | Mobile devices allow quick responses despite location, participate in social networking and maintain a variety of communication |
| **Development Tools** | Xcode is apple’s IDE and offers everything for develop in Mac environment.  *Swift* is the main language for Apple apps, others are:  *OBJ-C, C#+MAUI and JavaScript.* | Languages consist of *C, Rust and Assembly languages.* IDE tools consist of *Netbeans, Eclipse, Atom and Microsoft Visual Studio* | Lack of built-in software, downloads include *Eclipse, PyCharm and MS Visual Studio.*  Languages consist of *C, C++, Java, Python and Visual Basic.* | Visual studios for cross-platform development. Languages will consist of *JavaScript, C# and HTML* |

## 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**: My recommendation would be *Windows* 64-bit OS. This allows access to several IDE’s and languages along with up-to-date user-friendly software.
2. **Operating Systems Architectures**: Due to *Windows* being user-friendly the GUI is familiar to most allowing users to find storage with ease, software is abundant and one of the best OS when considering games.
3. **Storage Management**: The storage management system within *Windows* is great because files long dormant are freed up, and the ability to partition the file system allows for customization options. In future upgrades cloud storage can also be considered if the need arises.
4. **Memory Management**: The memory system with the 64-bit will take the physical memory of say 64GB and allocate 32GB to the virtual address. This allows for more processing power and as demand could increase the *Windows* OS can manage this without having to upgrade frequently.
5. **Distributed Systems and Networks**: With *Windows* being the product of Microsoft and Microsoft being responsible for the PC, along with systems within both the gaming ecosystem and mobile devices, their influence is far and wide and would have no issues communicating with other platforms.
6. **Security**: Security is a must-have for the client. *Windows* offers defender and firewall, plus cumulative updates to protect from malware and other malicious software. The OS does have slightly more issues with security than the other OS’s but the pros outweigh the cons and *Windows* is a secure OS.