

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 | 05/25/2025 | Connor Mura | Creation of document |
| 2.0 | 06/08/2025 | Connor Mura | Document Updated |
| 3.0 | 06/22/2025 | Connor Mura | Completion of 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, serving multiple platforms, that is based on their current game: Draw It or Lose It. Draw It or Lose It is similar to a 1980s game show called Win, Lose or Draw. The game consists of four, one-minute rounds where players compete to guess what is being drawn – or in this case – based on picture clues rendered from a stock photo library. The drawings are rendered in a 30-second timeframe, in which the team attempts to guess. Upon failure – the remining teams will have 15-seconds to try and “solve” the picture.

## Requirements

* The game must accommodate one or more teams of multiple players
* Both game and team names must be unique and allow users to check and see if a name is taken
* Only one instance of the game may exist in memory at any given time
* The game must be available on multiple platforms

## [Design Constraints](#_2et92p0)

* The game must be limited to one instance
* There needs to be a way to set, track, and search for team names
* There needs to be a way to not allow duplicate names
* The game needs to be able to accommodate multiple platforms so it needs to be compliant with any company policies that would hinder that
* The game needs to be web-based so it also needs to be secure and compliant with any laws/policies
* The game needs to allow for either one or more than one team to play, meaning something needs to address the failure-round guessing

## [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)

Provided below is a UML class diagram for the game. The class Program Driver is where the program is run/accessed. The Singleton Tester class is simply used to test the Singleton pattern deployed in the Game Service class, ensuring that only one instance of the game can exist in memory at any given time. The Game Service class manages the objects from the Game class. The Game class manages the teams that are playing the game. The Team class manages a single team and its players. The Player class manages a player of the game. The Entity class acts as a parent class to Game, Team, and Player.

This application demonstrates the use of Object-Oriented-Programming principles and is shown in the UML below. The four main principles of OOP are encapsulation, inheritance, polymorphism, and abstraction. We see encapsulation in our classes, as they manage their private variables only within the class. We see inheritance with the Entity class and its child classes; Game, Team, and Player – as each inherit from the Entity class. Polymorphism is seen in those classes when the toString method is called, as each child class overrides the Entity classes method. Abstraction is seen in our Entity class, as it is an abstract class only serving as a parent for other 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.

The cell-based format has been replaced with something I feel is easier to read and better for the purposes of this document due to the increased length required for each cell.

**Development Requirements:**

**Server Side –**

**Mac:** Mac is generally good for server configuration, reliability, and is generally developer friendly. The downside is the cost, as the licensing fees associated may not be as cost effective as other options.

**Linux:** While good for accessibility and server configuration – the greatest benefit to Linux is the fact that it’s open-source. This makes it far cheaper to use, especially when scaling a project. The downside being that Linux is not as user-friendly and will require more technical ability to work with.

**Windows:** Windows is very common and very user friendly. Microsoft also offers more support than other operating platforms. Like Mac, the downside is the increased cost associated with licensing fees.

**Mobile Devices:** Device specifications vary far more than with other operating platforms and are limited as mobile devices are not used nearly as often as the other platforms.

**Client Side –**

**Mac:** Mac can be more expensive but due to the popularity of the iPhone, it is very user friendly. Mac also offers consistent experiences across different browsers. Outside of the iPhone however, Mac isn’t as common as Windows.

**Linux:** While being the less expensive option, Linux is far harder to use for users without experience. Linux does however offer a higher degree of customization to users, which may be an important factor for some customers.

**Windows:** Windows is far more common than both Mac and Linux, and due to this familiarity is very user friendly. Windows also offers the furthest reach out of any of the platforms, due to this popularity. However, Windows can have browser inconsistencies across versions, leading to more required testing.

**Mobile Devices:** Mobile Devices are very common, leading to a large reach. They are also very user friendly, and with the appropriate design can offer great user experience. The issue with mobile devices is the amount of variance between different manufacturers and operating systems as well as the difference in user input when compared to the other options.

**Development Tools –**

**Mac:** Xcode is common for Swift, Objective-C, and other similar languages. VS Code is also popular. JetBrains IDEs are common for languages like Java and Python. Nova is also popular. There may be higher licensing fees associated with these IDEs.

**Linux:** JetBrains IDEs, VS Code, and Eclipse are popular and common languages like C, C+, and Python is also popular. Linux also has a variety of open-source IDEs available, but Linux can be harder to use.

**Windows:** JetBrains IDEs, Visual Studio, and Eclipse are popular and common languages like C#, C++, and Python are popular. There may be higher licensing fees associated with these IDEs.

**Mobile Devices:** Xcode for Swift and React Native for JavaScript are popular. The main downside is there are platform specific languages that may require developers with specialized knowledge.

## 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**: I believe Windows is the best operating platform to choose for this project. Windows is very easy to use and access because of how common it is. Widows also has a variety of programming languages and IDEs to choose from. As mentioned before, Windows is also very common and therefore will give the game the best chance for success. Windows also has a large community of developers and therefore offers a large amount of gaming engines and libraries as well as many gaming APIs.
2. **Operating Systems Architectures**: Windows offers both 32-bit and 64-bit architectures, and there are multiple versions of Windows as well. Due to the scope of the project as well as the goals, I’d recommend going with 64-bit Windows 11 due to the support and options that will be available if needed, the increased RAM, improved performance, and security features.
3. **Storage Management**: Windows supports NTFS, which offers additional security features, data integrity and recovery, file-level encryption and compression along with overall support for large files and will help with managing a game this large.
4. **Memory Management**: Windows uses a combination of RAM and hard drive space and automatically manages memory allocation.
5. **Distributed Systems and Networks**: To accomplish the goals of this project, I recommend using client-server architecture on a cloud platform. This will offer scalability for the growth of the project, a wider reach allowing users on other platforms to play, and cost efficiency when compared to other options.
6. **Security**: Windows offers many security measures like Windows Sandbox and Windows Defender. Windows Sandbox keeps running applications like games from affecting the rest of the system, and Windows Defender offers features like antivirus protection and firewall management to protect users.