

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 | 05/21/2023 | Matt Tranchina | Brief description and UML explanation. |

**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)

Draw It or Lose It is an Android based game where teams compete to guess what is being drawn. The application uses a large library of stock drawings to use as clues. Each game has four rounds lasting one minute each. Each drawing is rendered at a steady rate and is completed at the 30 second mark. If the team does not guess before time expires, the other team has 15 seconds to guess and solve the puzzle. The Gaming Room wishes this to be a web-based version of the Android app but has no knowledge of how to set up the environment.

## 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.*

## [Design Constraints](#_2et92p0)

This is not a new project from the ground up, but rather a “rebuild” of a current design. Currently the application only runs on Android based operating systems. In order for it to be a web-based application, alterations must be made. Cross-browser and cross-platform testing would be required. Budget restrictions may be a factor.

## [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 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.

The illustration above is a diagram of the code’s implementation. The Program Driver is where the source of the commands take place. It uses the Singleton Tester class to ensure that only one of each class is instantiated. The Game Service class creates and holds information for games, players, and teams. It can also call the Singleton Tester to ensure that only one game, player ID, and team ID exist at one time. The Game, Team, and Player class all inherit the traits of Entity, for ease and cleaner code since each of these require ID and a name. The Game class can add a team to a Game that was created in the Game Service and ensure that the team is unique. The Team class can add players to a team, and the Player class allows the creation of a player and ensures that each player is unique. Team and player names and IDs are set to private, so once they are created, they cannot be altered.

## [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 own paragraph response covering the indicated information.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Extremely user friendly, especially if using Mac computers. Built in sharing for File Server, Caching Server, and Time Machine Server. Can support other operating systems like Windows and Linux. | Linux servers are built on the Linux open-source operating system, meaning low cost. Linux is the most popular server for their stability, security, and flexibility. Allows full root access for complete control. | Windows servers are widely used and user friendly. Servers are stable, secure, and can work with multiple platforms. Supports web hosting but requires licensing costs that must be renewed. | Mobile devices such as Android and Apple can host basic web applications but do not support multiple user access. If a website is relatively basic, mobile devices are a cheaper alternative to expensive servers. |
| **Client Side** | Mac computers and laptops are expensive, and Mac development requires Mac hardware. Hardware is non-upgradeable. Hardware has Xcode built in but does support third-party development tools. Ideal for apps or low graphic games. | Linux is a versatile operating system. The top 500 supercomputers all use Linux. Linux is open-sourced, so operating system is free. Primarily written in C and C++, can support other languages such as Java, JavaScript, and Python. | Less expensive than Mac but Windows systems still comes at a cost. Can work simultaneously with Linux. Supports multiple languages with Windows SDK available, providing the latest headers, libraries, metadata, and tools. | Mobile devices are intended for user-end, not for development. Apple devices primarily used with Xcode IDE and supports Objective-C and Swift languages. Android SDK can support more common languages such as Java, C, C++, and Go. |
| **Development Tools** | Xcode is made by and for macOS for development in macOS and iOS. Can support third party IDEs like Visual Studio Code. Additional developer tools and APIs can be bought and downloaded through the Apple store. | Linux has multiple development tools and IDEs available; all are free. Can support Visual Studio Code for C and C++ languages and Eclipse for Java and JavaScript with C/C++ integrations. | Windows supports multiple languages and many IDEs. Most popular is Visual Studio with C++, Windows kernel is developed mostly in C. Many IDEs are free for basic integrations, with fees for more advanced features. | Android Studio is the official IDE for Google’s Android OS. Can be developed with other IDEs such as Visual Studio or Eclipse.  Apple is exclusively written in Xcode IDE using Objective-C and Swift language. |

## 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**: <Recommend an appropriate operating platform that will allow The Gaming Room to expand Draw It or Lose It to other computing environments.>
2. **Operating Systems Architectures**: <Describe the details of the chosen operating platform architectures.>
3. **Storage Management**: <Identify an appropriate storage management system to be used with the recommended operating platform.>
4. **Memory Management**: <Explain how the recommended operating platform uses memory management techniques for the Draw It or Lose It software.>
5. **Distributed Systems and Networks**: <Knowing that the client would like Draw It or Lose It to communicate between various platforms, explain how this may be accomplished with distributed software and the network that connects the devices. Consider the dependencies between the components within the distributed systems and networks (connectivity, outages, and so on).>
6. **Security**: <Security is a must-have for the client. Explain how to protect user information on and between various platforms. Consider the user protection and security capabilities of the recommended operating platform.>