

Draw It or Lose It – Web Version

# **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 | 01/21/2023 | Ty Simpson | 1st version of Software Design |
| 2.0 | 02/03/2023 | Ty Simpson | Completed Evaluation Section |
| 3.0 | 02/18/2023 | Ty Simpson | Completed Recommendation 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)

Our client, The Gaming Room, has asked for our assistance in setting up the game environment for building a web version of their Android-only game *Draw It or Lose It*. The game should consist of one or teams, with multiple players on each team. The name of each game and team should be unique. A game should have only one instance in memory at any given time. This can be achieved through object-oriented programming and using singleton and iterative design patterns.

## Requirements

* One or more teams to a game
* Multiple players per team
* Unique game and team names
* One instance of a game in memory at any given time
* Web-based

## [Design Constraints](#_2et92p0)

* To be compatible with multiple platforms, different development kits will be needed
* Game and team names will need to be unique – this can be solved iterating through a list of created names and using iteration, compare the user input to names that already exist
* Teams, Games, and Players must be assigned a unique ID per instance; limits game instance to one

## [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 ProgramDriver is the main system – the ProgramDriver calls the SingletonTester to confirm that only one instance is running of GameService. The Entity class is the Parent class – and Game, Team, and Player all inherit from it. A team must consist of players, and a game must consist of teams. GameService must have games. The singleton design pattern is displayed in this diagram, which means, at any give time, only one instance of GameService can exist.

**"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 own paragraph response covering the indicated information.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac server-side development is possible. However, licensing is expensive and viewed as a niche market – meaning limited support and scalability. | Linux is well equipped for the server-side. It is open source, which means zero licensing costs and full customization. However, Support costs are not free and licensing fees for these services may cut savings. | Windows has a well-established history for server side. They are secure, easy to use, and come with a bunch of support. Licensing is expensive. | While the cost is low, mobile devices lack the power and the capacity to be effective servers. |
| **Client Side** | Development on the Mac side requires knowledge of the Swift language. | There is a steep learning curve when developing on Linux – you will need experienced Linux developers with knowledge in Python. | Developing on Windows requires knowledge in C# and the .NET Framework. | Developing for mobile applications would require developers familiar with mobile. The Gaming Room is fortunately set up well for this, as the app was originally developed for Android. For iPhone, familiarity with Swift and/or Objective-C is needed. |
| **Development Tools** | Most applications can be built using Apple’s IDE Xcode. | IntelliJ IDEA is an excellent IDE for Linux environments. Developing for web applications would require purchase of the Ultimate license. | The Visual Studio IDE is a great option for developing on Windows. | As the app is already built for Android, I will be focusing on options for Apple’s mobile iOS. Piggybacking off of the Mac evaluation, you could use Xcode, writing in Swift and Objective-C. You could port the android app, but you may be exchanging the convenience and costs with the ability to code efficiently for the native 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**: A cloud-based Linux server instance is recommended.
2. **Operating Systems Architectures**: Linux is open source and can be customized to your needs. Linux is highly secure and stable.
3. **Storage Management**: Storage can be set up on the cloud server instance. It is recommended to use SDD for faster performance.
4. **Memory Management**: Since the server is set up in the cloud, memory is scalable and elastic – which means that resources grow and shrink according to the business needs.
5. **Distributed Systems and Networks**: Cloud servers are highly available and fault tolerant, which means server interruption is minimal and in the event of an outage in a cloud region or availability zone, the cloud server can be moved to another to keep the lights on.
6. **Security**: Utilize the principle of least privilege with role-based security. Draft a security policy document to best determine what level of access each role has.