

# **CS 230 Project Software Design Template**

Version 1.0

## 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 | 09/17/21 | Adam Conger | Initial markups and comments |

**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 staff at The Gaming Room want to develop a game application that is similar to Win, Lose or Draw. The game will have one or more teams, potentially with multiple players. Game and team names will be unique, and for memory saving purposes, only one instance of the game will exist within the memory.

## [Design Constraints](#_2et92p0)

We will have to make sure that our code is aligned with creating an single instance of a class to save memory. (We can use singleton in the Java code). We also want to make sure we preserve the unique values that enter through the Game class. (We can use an iterator here).

## [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)

ProgramDriver and SingletonTester run the main code and also test for the unique instance we created within GameService. That class hosts the singleton we created. Our Entity class declares our values for Entity. The Game, Team and Player classes inherit from Entity and declare their respective values.

**"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** | **T-Second Choice**  Mac is widely supported and has a few network applications like Jamf that can make hosting Mac devices simple. Flexible terminal. | **Fourth Choice**  Open source and can be worked in a community setting, it’s the least accessible for inexperienced developers. Cost friendly. | **First Choice**  Windows has plenty of network applications that make hosting large servers easy. Most software available in comparison. | **T-Second Choice**  Mobile Devices occupy a particular market space so it’s almost a mistake not to support this area. Best if the server doesn’t move and in one area. |
| **Client Side** | Here are some of the client-side software development considerations regarding multiple clients for Mac: moderate amount of expertise, moderate time needed. Cost similar to Windows | Here are some of the client-side software development considerations regarding multiple clients for Linux: high amount of expertise, high time needed. Minimum cost. | Here are some of the client-side software development considerations regarding multiple clients for Windows: least amount of expertise, least time needed. Cost similar to Mac. | Here are some of the client-side software development considerations regarding multiple clients for Mobile Devices: flexibility with updates at any place, but difficult to implement. |
| **Development Tools** | Here are some of the development tools regarding Mac: Python, Java, Ruby on Rails for languages; Eclipse, PyCharm for IDE | Here are some of the development tools regarding Linux: Python, Java, C, C++ for languages; Eclipse, PyCharm or many other IDEs for Linux | Here are some of the development tools regarding Windows: Python, Java, C, C++, VB for languages; Eclipse, Vis Studio, PyCharm for IDE | Here are some of the development tools regarding Mobile Devices: Python, Java, Ruby on Rails for languages; Eclipse, PyCharm for IDE |

## 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**: The appropriate platform for The Gaming Room would be Windows. Eventually, they would be able to expand Draw It or Lose It to mobile devices.
2. **Operating Systems Architectures**: Windows would allow for user mode and kernel mode, giving users access to applications but also the kernel access if they need it. Kernel mode keeps all of the protected and private functions of the hardware from regular users.
3. **Storage Management**: Storage management options can include local storage for the end user (saving local data) and remote server storage options (having a server room remotely that runs the online player servers, saves gamer account info, match history).
4. **Memory Management**: Windows operating platforms have very flexible memory management methods. Windows platforms will rely on 8gb – 16gb memory for the fastest experience for most users with Draw It or Lose It.
5. **Distributed Systems and Networks**: The biggest aid in cross platform communication will be remote servers running storage management and memory management. A remote server room that is efficiently built can handle many processes and redirect users to different ‘realms’ if they experience any outages.
6. **Security**: Windows platforms will definitely need security measures; so encrypted user information will be necessary. Users could also consider implementing a 2-factor authentication measure for their account, or having a verified email to register the account.