

Draw It or Lose It

# **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 | 01/29/22 | Vincent Longano | Document Created |

**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 wants a web-based program that runs a game where users have to guess what image is being rendered. There will be one or more uniquely named teams, each with multiple uniquely named users. During a round, a start state in the program will start rendering an image from a library of stock images. The active team will have 30 seconds to guess the image rendered. If the puzzle is not solved, the remaining teams will each get one chance to solve the puzzle within 15 seconds.

This program will utilize a game-service class. There may only be one instance of the game service in memory at a time. The service will handle the creation of games, players, and teams. An entity object will be the base object for the games, players and teams.

## [Design Constraints](#_2et92p0)

The game must be ran from a single game service in order to be able to keep track of object instances and keep consistency between all players. This will be done by implementing the singleton pattern.

Each team name and player name MUST be unique. This will be done using an iterator pattern to check to see if a name already exists.

## [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 GameService class contains a private constructor to keep it singleton. The GameService will be able to create multiple Game objects. Each Game object will be able to add multiple Team objects, and each Team object will be able to add multiple Player objects. The Game, Team, and Player object all extend the Entity object.

**"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 is generally more expensive for the hardware. The OS offers nothing in particular to cater as a server. Since it is Unix based, it may have better terminal functionality than say windows. | Linux has a lot less background processes that could slow it down when number of users is scaled up. Since it is open source, it could be harder to maintain security. Linux probably has the most flexibility for what the server needs to incorporate. | Windows is in the middle as far as price since the hardware isn’t comparably expensive however licensing costs add up fast. Windows’ gigantic market share makes it highly targetable to cyber attacks due to a large userbase which could be bad news for a server. Lots of background processes could also slow down the server as more users connect. | It is safe to say mobile is not the choice to host the server. While its not inherently a design flaw of the OS itself, the hardware is simply not made for it. Modern mobile devices could probably host a few connections no problem, but as soon as you scale it will slow down. There are also other factors such as usually being limited to wireless network communications, which is not only slower but less secure than hard wire connections. |
| **Client Side** | Mac supports most popular platforms right now such as Java. It shouldn’t take too much effort for a port and could pay off significantly considering the rise in popularity of Macs especially within the demographic of who would play this game. Macs also have reliable security to reduce cheating and hacking attempts. | Linux would be a little tricky to effectively deploy. Being open source and having a lot of different versions could incur compatibility problems. Another problem would be security. The user base is probably too small to justify the costs as well. | Though there are many downsides to windows, this should be the primary version along with mobile based on user base and familiarity alone. Though windows itself has almost too much security features built in, it still is a prime target for cyber attacks so that is something to keep in mind. | Like windows, mobile should be a primary focus just because of the user base and access. Since most people will have this in their pocket wherever they go. There will be a few things to consider such as connection reliability with possible dead zones. Modern mobile devices should have no problem running this software. |
| **Development Tools** | Based on its growing popularity, Mac has support from a lot of the big IDEs such as Eclipse and supports high level languages. | Linux has plenty to offer as far as development tools. It also supports almost every language. These include Java and Eclipse. | Windows might have the best direct proprietary support for the IDEs and languages needed to build this software. | Mobile has little to no actual development tools. You may be able to indirectly support the development through a mobile device but otherwise it would just not be worth it |

## 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 would recommend windows server for the operating platform. Between supporting multiple languages and many developers having a lot of experience with it, It is very cost effective for the relatively simple program.

**2. Operating Systems Architectures**: The architecture of windows is generally a user-friendly interface where the server can be controlled through the command prompt.

**3. Storage Management**: it may be smart to use cloud storage for this app. The storage needed should be relatively cheap. It may also be wise for the client side to cache the pictures before rendering to ensure all users see the picture rendering at the same time despite differences in connection speed.

**4. Memory Management**: If the game is written in java, memory management will be automatic. All necessary work will be done between java and the operating system. With many different players being able to connect, It may be wise to invoke destructors on the player and team objects when disconnections are detected in order to keep the amount of objects at a reasonable quantity.

**5. Distributed Systems and Networks**: The use of an API such as HTTP would be the best way to connect all the platforms to each other since different clients can be added but the code stays the same throughout.

**6. Security**: A user authentication system would be the best line of defense against unwanted connection. To go one step further, users could have to make an account and have an email address verified. This would deter spam accounts and make sure only human users are connecting to the server. Windows has some intrinsic security build in to protect from hacking, but there should be checks and balances within the code that double check the information sent back and fourth to ensure cheating or hacking attempts aren’t happening.