

<Name-of-Software-Application>

# **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 | 11/14/21 | Raymond J Gibson | Initial development |

**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 is looking to implement a game similar to one in a 1980’s television show Win, Lose, or Draw. The program will draw from a large library of stock images. Drawings will be fully rendered on the screen in 30 seconds. The first to guess what it is correctly win.

## [Design Constraints](#_2et92p0)

1. Games must have the ability to have one or more teams involved.
2. Every team must be able to have multiple players
3. All game and team names must be unique
4. Only one instance of the game can exist in memory at one time.

## [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)

Below is the proposed UML model. We have a base class Entity that is inherited by our Games, Team, and Player classes. Entity provides our basic attributes for its child classes. Here we initialize the ID, and name, and can retrieve both of those attributes as well. Game, Team, and Player are all very similar. They store constructors, and Game and team store lists of already created names/IDs. This allows us to compare new Game or Team instances to the ones that already exist and keep us from duplicating them. Finally, we have our GameService class. This manages the singular instantiation of the games in memory. From here we can get/add games.

**"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** | We would not recommend Mac as a server OS, it is not often used in the industry and does not meet all requirements. | Linux is a great OS to user on the server side. It is free and open source, meaning it is highly customizable and configurable. That said it is slightly harder to setup and maintain and requires a bit more expertise to admin then say windows. | Windows servers are a bit more beginner friendly than Linux, but they also come with a licensing fee. They may also come with support from Microsoft which can be nice if you need it. | Mobile devices do not host web servers. |
| **Client Side** | Designing and maintaining web based architecture is possible on Mac. Many people like to use Mac as much of the design aspect of web apps is done using graphics programs that Mac is particularly well suited for. From their one can find many programs for writing code in a multitude of languages. | Linux is probably not the OS of choice for doing coding and design for the client side of web apps. It is certainly possible using something like VIM but most people want a modern IDE for whatever language they are using. | Windows strikes a good balance here as there is an IDE for all the popular languages available and graphics and design software runs well on windows as well. | Web app design and coding can be done for IOS and android on either Windows or Mac. |
| **Development Tools** | JAVA, Python, html, css, docker, homebrew, flask, Django, SQL | JAVA, python, html, css, Apache, PHP, MySQL | JAVA, C++, Python, MSSQL, html, css, | Xcode, React, JavaScript, swift, objective-C, Visual Basic |

## 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**: We absolutely recommend Linux for hosting the game as it runs on most servers and will be easily portable and scalable.
2. **Operating Systems Architectures**: Linux consist of several parts in its architecture. At the lowest level is the Kernel. This core section of the OS, it takes care of the most important tasks and coordinates between different services, what memory or processing power they need. You can think of the Kernel as the air traffic controller of the system.
3. **Storage Management**: Linux has a built-in storage management system called SSM. We can use this to leverage any on site hardware we would want to stand up. More likely we would be doing this in the cloud however, AWS has several Linux options that can host various types of servers and we can really choose anything that we’d like.
4. **Memory Management**: The Linux Kernel will take care of memory management for us, allocating memory from the stack or heap as necessary. One nice feature of the Linux Kernel is that is also uses virtual memory. This virtual memory can be much larger than what is actually stored on a disk. Linux does this by allocating pointers in the virtual memory and then moving those pointers to real memory only when they are called or instantiated, saving real physical memory space for actual data and not filling it up with a ton of pointers.
5. **Distributed Systems and Networks**: Our clients best bet is to host the majority of the system in the cloud, probably using AWS. We can setup a linux server for them for their code, use RDS to setup their database, and then run the whole thing off EC2. This will make real time scaling easy. We can also run a local backup of the RDS instance in a physical location as replication service. This will help us if there is ever a disaster we will have a backup of our data, and we will also have quick access to it for running any analytics we may want to run. We don’t need to back up everything but player scores log in/out records, device statistics etc. would help us analyze our player base and make better business decisions.
6. **Security**: Linux is very secure for a whole host of reasons. One it is not Windows! Windows by virtue of being the biggest system is most susceptible to malware and viruses. The next is that the Linux Kernel is very robust, attacks against it are not easy to do. Linux has very powerful role separation guarded at the lowest levels of the OS, to keep users who need access from those who do not. There are also quite a few third party security suites just for Linux. Linux has also been used for quite a long time and there are a lot of industry best practices that a developer can follow to prevent vulnerabilities or backdoors.