

The Gaming Room

# **CS 230 Project Software Design Template**

Version 1.2

## 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 | 01/26/2024 | Jacob Englund | Changes to the executive summary, design constraints, and domain model sections. |
| 1.1 | 02/09/2024 | Jacob Englund | Added to the evaluation table for server side, client side, and development tools |
| 1.2 | 02/21/2024 | Jacob Englund | Added to the 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)

The Gaming Room is trying to create a web-based game based on the 1980s television show Win, Lose, or Draw. This game will be available on multiple platforms. The game requires multiple teams to go head-to-head in a guessing game. A team will guess what an image is within 15 seconds, but if they are wrong, another team will get the chance to guess. If a team guesses correctly, they get points. The team with the highest score wins.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

## [Design Constraints](#_2et92p0)

* Only one instance of the game can be running at one time
* Teams must be setup before the game starts
* Must be able to run on multiple platforms
* Teams should have multiple players
* No name duplicates for teams or players

## [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 domain model below shows that the entity class is the parent of three classes: Game, Team, and Player. This means that Game, Team, and Player all inherit the functions and variables that Entity creates. Game is also the parent of GameSerive. This means GameService inherits the functions and variables created by Entity and Game. The diagram also shows that ProgramDriver uses SingeltonTester. The three classes that are children of Entity are each connected with a “0..\*”. This symbol means that there can be 0 to several attributes shared with one another.

**"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 easy to access and user friendly. However, Mac is expensive to maintain. It is also dependent on its own software making third-party software difficult to use on Mac. Semi-regular updates for security.> | <Linux is very inexpensive. It is difficult to navigate and requires shell commands to navigate. Does not support all third-party software. Open source making security an issue.> | <Windows has a user-friendly interface. License for Windows is expensive. Extensive support for third-party software. Regular updates for security.> | <Low processing power. Operating systems vary from device to device. IOS depends on its own software making third party software difficult to use. Android allows access to third party software.> |
| **Client Side** | <OS is built on the Swift language. Swift, like any other language, requires a small amount of time to learn. Expertise is not necessary due to how user friendly it is. Equipment can be expensive.> | <Linux requires a lot of expertise and time to learn because of its lack of user friendliness.> | <Licenses are expensive. Expertise is not necessary. Easy to understand making time to learn low.> | <Expertise needed to implement mobile devices. Some devices can be expensive. Time to learn is based on device’s operating system.> |
| **Development Tools** | <Mac is built on a language created by Apple called Swift. It also uses languages like HTML, JavaScript, and C. Applications are made with Apple’s own tool called Xcode.> | <Linux uses languages like Python, C, and C++. IDEs for Linux include Visual Studio and Atom. | <Windows use languages like C#, C++, and Java. Most applications for Windows are developed in Visual Studio.> | <iOS devices use the apple language Swift and are developed on Apple’s own tool Xcode. Android devices use HTML, Java, and C++ and can be created in Visual Studio and Eclipse.> |

## 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 operating system I would choose for the game would be Linux. While Linux is difficult to learn, once started, I believe it will be the easiest to maintain. Also, because Linux is open source, it will be extremely easy to integrate other systems to offer cross-platform gaming.
2. **Operating Systems Architectures**: Linux uses a system called kernel. Kernel is the core of Linux in that it controls resources such as the CPU, memory, and file storage. Because Linux is open source, anything can be created for it. It is like a clump of clay waiting to be molded exactly how you want it.
3. **Storage Management**: Storage for this game will not need to be overly huge. Items that need to be stored will be simple things like account details and images when they are not in use. A simple server filled with hard drives will be more than enough for this game and will allow future expansion.
4. **Memory Management**: This game requires images to be immediately rendered for each round. Because of this, each image will need to be stored in the quick access storage called RAM. There are not an incredible number of images, but each game will require its own allocation. Because of this, a server with plenty of ram will be required. The benefit of using a server is that you can add or remove things as needed. If the game is extremely popular and needs more RAM, simply add more RAM and vice versa.
5. **Distributed Systems and Networks**: Servers are what bridge the gap between operating systems and consoles. These devices cannot communicate with one another due to software or hardware restrictions. This is where servers come in. They store all necessary information so the device can simply focus on displaying the game. This way the game can take up a small amount of space on the device, while at the same time, delivering the desired output.
6. **Security**: The best security method for securing a user’s account will be usernames and passwords. When a user first enters the game, they will be prompted to create an account. This account is then created on the server, so when the user returns, they will have all their data. Because this is a game, this is all that will be necessary. However, in more important situations, something like two-factor authentication can be set up so the user has an extra layer of security for their precious data.