

<Name-of-Software-Application>

# **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  1.1 | <07/10/23>  <08/11/23> | <Bryce Burroughs> | < Added recommendations and Requirements>  <Updated recommendations> |

**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 us to create an app that will function similar to the 1980s TV game show, Win Lose, or Draw. This game will have teams competing to guess what is being drawn on the screen.

## Requirements

Each drawing segment will last 30-seconds. The team guessing must guess within those 30 seconds or else every other team will have 15 seconds each to guess once. The app should have one or more teams involved. Each team will have multiple players. The game name and team names must be unique and check whether a name is in use when choosing names. Only one game may be ongoing at a time.

## [Design Constraints](#_2et92p0)

Should be compatible with all modern browsers, should function efficiently, should be able to handle many users, and should be secure with user data.

## [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)

Entity creates a relationship between the three subclasses: Player, Team, and Game. These classes inherit information from Entity. These three subclasses also are “has a” type classes. Game has a Team and GameService has a Game. If we look from left to right we will see each class “has a” of the class to it’s right. GameService has a Game, Game has a Team, etc.

**"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** | Upgradeable, somewhat common, and no susceptible to viruses. | Least common of all OSes means less compatibility. Secure, but most difficult to implement. | Very commonly used, means higher compatibility. | Lower specs than desktop, but higher portability in return. Wider reach to audience since most people have cellphones. |
| **Client Side** | Less common than Windows or Mobile, means more specific expertise is required when looking at development. Cost is similar to Windows. | Very niche OS means that the maximum amount of expertise would be required to develop for this OS. | Most common desktop OS, least amount of expertise required as such. | Most common OSes, but since it has less computing power will need to be incredibly efficient, therefore may be more expensive than Windows. |
| **Development Tools** | HTML/CSS/JavaScript/Swift Can run most languages. Could use tools such as notepad++ | VSC,Eclipse,Notepad++ Many languages supported, including HTML/CSS/JS/Ruby/Python | Easier to use than Linux, while also supporting the same IDEs and Languages as Linux. | Android and Swift languages can be ran on all three other OSes, many languages supported here also supported on Dekstop OSes. |

## 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**: In order to expand to other environments and expand scalability, I recommend that we build this on cloud-based platform. Something like AWS or GCP offer infrastructure to accommodate the expansion of the game while giving us flexibility, scaling, and ease of management. These platforms also allow for easy deployment across a wide range of OS and environments. For our use-case I would recommend AWS.
2. **Operating Systems Architectures**: A cloud-based platform will be able to support a variety of OS architectures. This will include Windows, Linux, and other OSes as well. Each of these architectures have their own strengths and weaknesses, but for the case of this project we will focus on Windows as it is the most common.
3. **Storage Management**: For storage management, we can use Amazon S3, which provides scalable object storage with high durability and availability. Game assets, user-generated content, and data can all be securely stored using S3. We can use Amazon RDS for consistent data storage, such as profiles, high scores, and game progress.
4. **Memory Management**: AWS’s cloud infrastructure handles memory management all on its own. It does this by managing memory allocation and deallocation. It also provides Elastic Load Balancing and Auto Scaling which dynamically manage resources based on demand.
5. **Distributed Systems and Networks**: To enable cross-play, a microservices architecture can be used. Each game can be separate microservice deployed on AWS and these services will communicate using APIs.
6. **Security**: AWS offers a host of security features to keep user information safe. These include AWS Key management, SSL/TLS protocols, and regular security audits.