

The Gaming Room

# **CS 230 Project Software Design**

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 | 4/25/2024 | Megan Zurawski | Software design changes |

[**Executive Summary**](#_sbfa50wo7nsh)

The Gaming Room app is aiming to develop a web-based game inspired by the 1980’s TV game show Win, Lose or Draw. This app is currently available only as an Android application named Draw it or Lose it. However, they lack the experience to set up the development and need assistance in streamlining the process. The vision for the new version of this game includes enabling gameplay for multiple teams with multiple players each, allowing unique team names, and expanding platform availability.

## [Design Constraints](#_2et92p0)

Gaming Room Design Constraint’s:

* The game should be web-based for easier accessibility.
* Gameplay should consist of four rounds, each lasting one minute, four minutes total.
* Drawings must be completed by the time limit mark.
* If a team fails to guess within the time limit, other teams get one additional guess within 15 seconds.
* The game should support one or more teams, each with multiple players at a single time.
* Team and game names must be unique to avoid conflicts.
* Only one instance of the game should exist in memory at a time, achieved through unique identifiers for games, teams, and players.

## [Domain Model](#_8h2ehzxfam4o)

The Entity class serves as the handler for Game, Team, and Player classes, because of how it takes on different characteristics. These classes reference each other, forming the main structure of the game. The ProgramDriver class is responsible for building the project to meet the client requirements, offering access to all created classes for when it is implemented. Additionally, the SingletonTester class ensures they are sticking to correct design constraints, allowing for multiple teams and players while maintaining a single unique game session.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Higher security and stability, better for hosting a web-based application. Has an advantage due to its many server capabilities. | More known for its reliability and lower cost overall. Has a more “open source” nature to it, making customization possible. May require more technical knowledge. | Offers simple integration across several devices using the same products and tools. Using this as an infrastructure allows much more ease. Has a higher cost than most. | Not as well known for being used for hosting a web-based application but still known. Offers ease with portability and accessibility but lacks the same server capabilities the others contain. |
| **Client Side** | Can support multiple client platforms. The cost will all depend on what is chosen for framework. Has great adaptability for the application across platforms. | Also supports multiple client platforms extremely well. Cost is also dependent on what tools are needed for specific customizations. | Can provide various developing tools and frameworks when building cross platform capability. Can run into issues if used on a non-windows platform, would need more testing. | Can support multiple platforms between IOS and Android mobile. Would need more expertise in handling mobile frameworks. Cost could include a licensing fee and testing on the devices themselves. |
| **Development Tools** | Programming development tools could include Xcode and licensing costs. These usually are included in Apple’s develop fees in whatever program is being used. | Involves open source language such as python or Ruby. Uses IDE’s such as VS Code usually cutting out the licensing costs. | Development tools used with windows is more frequently Visual Studio. Cost depends on which edition is being used and how much it is used overall. | Tools for mobile involve Android studio or Xcode for IOS. There are also cross platform frameworks like Flutter or React Native, which reduce time and cost. There still however could be licensing fees. |

## 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**: For an operating platform that supports expansion into multiple environments, I recommend utilizing a cloud-based platform. Cloud platforms offer more scalability/flexibility and are much easier to use across different operating systems and devices. I would recommend using a platform that has extensive support for multiple operating systems and their different infrastructures.
2. **Operating Systems Architectures**: In our cloud platform, the architecture would likely be based on a virtualized environment. This architecture allows for resource allocation and management across different machines running multiple operating systems. Containers can be utilized for application deployment, providing consistency and portability across different environments.
3. **Storage Management**: For storage management, a cloud-based object storage service like Azure Blob Storage would be good to use. These services offer scalable and durable storage solutions with high availability and reliability. These services also make sure that the game's data is securely stored and easily accessible across different platforms.
4. **Memory Management**: The cloud based operating platform uses memory management techniques like virtual memory to efficiently utilize system memory resources. Virtual memory allows the operating system to provide each process with a virtual view of the memory, allowing efficient multitasking and memory isolation.
5. **Distributed Systems and Networks**: To enable strong communication between various platforms, a distributed system architecture can be used with technologies like RESTful APIs or even message queues. The game's different servers can expose different APIs for the multiple devices to interact with, allowing smooth cross-platform communication. A networking infrastructure with failover mechanisms should be in place to ensure uninterrupted connectivity between devices, even in the event of network outages.
6. **Security**: Security measures should be implemented at various levels to protect user information between different platforms and the individual platform itself. This includes encrypting data when at rest or in transit, role-based access control to restrict unauthorized access to sensitive material, or perhaps regular security audits/updates to prevent potential vulnerabilities. The cloud platform offers built-in security features like identity and access management, network firewalls, and data encryption, which can be used to enhance the overall security of the game.