

<The Gaming Room>

# **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 | 01/22/2025 | Brandon Hornick | Completed questions. |
| 2.0 | 2/17/2025 | Brandon Hornick | Completed Module 7 Reflection |

**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)

<Write a summary to introduce the software design problem and present a solution. Be sure to provide the client with any critical information they must know in order to proceed with the process you are proposing.>

The client, The Gaming Room, wants to develop a web based game “Draw it or Lose it”. However, they do not know how to setup the environment. The game will consist of four one minute rounds, in which they must solve the puzzle. If the puzzle is not solved before time expiration, the remaining teams get one guess each to solve within a 15-second time limit.

## 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.>*

The key requirements of the game are:

* The game must have the ability to play with one or more teams
* Each team can hold multiple players
* A check to ensure no usernames are already in use or repeated
* Only one instance of the game can be running at one time.

## [Design Constraints](#_2et92p0)

<Identify the design constraints for developing the game application in a web-based distributed environment and explain the implications of the design constraints on application development.>

The design constraints of the game are:

* Live interaction:
  + This is a design contraint because latency of data transmission must be a focal point as the game is played live.
* User name repetition:
  + This is a design constraint because the program must ensure that each username is distinct to prevent issues with storing the player’s data.
* Compatibility:
  + The program must be compatible across multiple browsers and devices, meaning tests need to be performed to make sure the program is optimized for separate browsers and devices without visual or functional issues.

## [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 UML diagram below showcases connections between different components in the Gaming Room program. The types of connections are listed below:**

1. **ProgramDriver uses (association) SingletonTester**
2. **GameService and Game are associated. They have a 0 to many relationship, which means GameService can handle many Game objects.**
3. **Game and Team are associated. They have a 0 to many relationship, meaning each Game can handle many Team objects**
4. **Team and Player are associated with a 0 to many relationship, meaning Team can handle many Player objects**
5. **Game, Team, and Player inherit from Entity. This shows that Entity is the superclass, and allows the below classes to maintain cleaner code without repeating unnecessary lines.**

**There is also encapsulation shown in the diagram, specifically in the GameService class. This is shown by creating private methods that can only be altered by the ‘setter and getter’ functions.**

**"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** | Advantages: High performance, stability, and security  Weaknesses: Less commonly used, less support and resources | Advantages: Stability, security, and its open source nature  Disadvantages:  Steep learning curve, less commonly used | Advantages: Broad support, compatibility, and most commonly used  Weaknesses: Can have lower performance, higher licensing costs | Advantages: portable, energy efficient, can be setup easily  Weaknesses:  Limited processing power, less stable and reliable, limited support |
| **Client Side** | Advantages: high quality display, great performance, reliable  Weaknesses: Not as compatible as Windows, less customization | Advantages: Efficient even on lower resource devices, great community support  Weaknesses: Higher learning curve, limited support and compatibility. | Advantages: high compatibility, strong support, large community  Weaknesses: less secure, often slower devices | Advantages: Portable, most intuitive device for client, integration to other phone apps  Weaknesses:  Slower and weaker than PC, limited screen size |
| **Development Tools** | Languages: Swift, Objective-C, Python, JavaScript, C/C++  IDE: Visual Studio, Comes with Xcode, PyCharm terminal | Languages: Python, C++, JavaScript, Java, Golang  IDE: Visual Studio Code, Eclipse, Clang | Languages: C#, C++, JavaScript, Python, SQL Powershell, Java  IDE: Visual Studio, PyCharm, SSMS | Languages: Swift, Kotlin, JavaScript, Objective-C  IDE: Android Studio, Xcode, Visual Studio Code, Expo |

## 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 recommend MacOS to deploy the Gaming Room application due to its strength in reliability and processing strength. A language like Python can be used to ensure cross platform compatibility, and can be used across all major platforms without any issues.
2. **Operating Systems Architectures**: MacOS is a Unix based system which creates a strong and secure foundation for the application. MacOS is known for its high performance and will be great for this application’s development.
3. **Storage Management**: SQLite is a great choice for storage as SQLite comes preinstalled with Python. This is a local storage system, but if the game progresses on a further scale, a cloud storage system may be desirable.
4. **Memory Management**: MacOS uses a virtual memory system utilizing RAM and memory for performance.
5. **Distributed Systems and Networks**: Using frameworks like Flask and Django we can create APIs that allow communication between multiple platforms. The application can also be deployed on a cloud system to allow multiple platforms access. To handle connectivity issues, tools like RabbitMQ, which holds data until the receiving end is ready to process the data, can be used.
6. **Security**: Encryption of things like passwords and usernames can be used to protect user information, especially when data is transferred through platforms.