

<Game Room>

# **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 | 09/18/22 | B. Isaac Medina | Added a class “Entity” which ties the other classes together for use in one system. |

**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 vision for this project is that a server be built and that this game will work running on multiple devices with multiple teams participating all at once. With that in mind, I would advise that the development team fix issues with validation of each instance as I know they do exist. I also advise that compatibility be thoroughly tested across all platforms, as this could cause major performance issues later if not thoroughly worked out now.

## [Design Constraints](#_2et92p0)

Remain within the realm of what is possible today. We aren’t designing anything that hasn’t already been made, at least from a technical vantage point. Be sure that the design remains appropriate for cross platform integration.

## [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)

In this diagram, Inheritance is a prevalent principle. More than half of this system is inherited from some other class starting with *Entity* at the beginning. GameService is the main working system and from it we are able to control all of the rest of our program. The other classes just define, run, and validate the team aspects of the game.

**"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** | N/A  MacOS servers have been discontinued. | Linux is one of the most lightweight and widely used server operating systems, but you’d have to have a team that really knows what they’re doing. | Windows servers are widely used especially for small servers and it’s learning curve is not as steep as Linux. Overall recommend this as a starter. | N/A |
| **Client Side** | Support for Mac systems should be pretty straight forward. Shouldn’t be too costly, or difficult to find someone who has expertise in mac systems who could build a native mac system. | I would like to assume that the similarities between Linux and Mac would enable the team to hire the same dev that did the Mac client to do this one too. | Windows seems to be straight forward. If you can get it on the internet, you can distribute it to the world! Might face issues with cache as time goes on. May need to find an efficient dump system for that to keep the experience smooth. | Both Apple and Android would be the main focus. Two different approaches to the design process could cause some hiccup for the team. May want to focus on only one platform at least at to start. |
| **Development Tools** | Swift, Xcode, NetBeans | Vim, Visual Studio Code, Seamonkey | Java, C++, VS 2022, intelliJ | Swift, Xcode, Kotlin, Android Studio |

## 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 place to start would be Windows Server 2022. It allows for dependable development support across almost all platforms, it is dependable, and quite the norm to use so it wouldn’t be a hindrance for the team.
2. **Operating Systems Architectures**: 32-bit should be sufficient although 64-bit may be already supported due to modern hardware.
3. **Storage Management**: As server hosted support is not functional at this stage, the most basic of storage on a home system is appropriate.
4. **Memory Management**: As the client will likely be programmed in C on a Windows system, memory management will be a default in the code.
5. **Distributed Systems and Networks**: LAN communication for gaming on a single network is very common. WAN support may be an appropriate approach at this early stage as well. Once full server support is up and running, the main system will use a REST API to communicate with client systems.
6. **Security**: Security is all over Windows core systems but it is would be a good idea to pay for services like MacAfee or Trend Micro to further secure the system. Security testing will need to be a major part of the development process as well. I would also recommend that a team be employed to monitor the security of the system as well. The client needs to rest assured that their system Is secure after the download and during use of the application.