

The Gaming Room

# **CS 230 Project Software Design Template**

Version 3.0

## 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 |
| --- | --- | --- | --- |
| 3.0 | 06/19/2022 | Cody Jurrens | Additional information was added to better understand the requirements and layout of the program. |

**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 client Gaming Room would like us to design a program that is a web-based version of their game draw it or lose it. The client has asked us to have some specific requirements for the game including: multiple teams, multiple players per team, only allow unique name so they aren’t used more than once, and only allow one game to run at a time do to memory allotment.

## [Design Constraints](#_2et92p0)

* The game must be written in Java or Python so it can be fully functional as a web based program.
* The game must be written so that there can be multiple teams with multiple different players per team.
* The game must be written so that only one instance can run at a time so there is low memory allotment, and the game continues to run smoothly in the web based environment.
* Must use a singleton pattern to check for unique team names and player names so there cannot be two of the same in each instance.

## [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 game, team and player classes all inherit from the higher Entity class. This will make sure that all of the attributes from the other classes only has to be written once so we can use the different classes for other projects or games that the client may want in the future. The GameService, Game, Team, and Player classes are all connected. This is what we call a zero to many because it can use as many attributes or instances from each other as they need to. The only two that pull from each other is the SingletonTester which pulls its information from the ProgramDriver to make sure the names and teams stay unique.

**"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** | You can rent a server from apple for around $20 which would be inexpensive for a game as well as cloud storage but Mac is far less popular for gaming so there might not be as many clients. | Linux would be a great option for cost as well because almost everything on there is free or close to it for your own server. It has many different distributions for low memory computers so it could run on many different machines. The problem would be that Linux could crash if you aren’t familiar with it and its also less popular for gaming. | Windows servers are quite a bit more expensive, but they give a range of options on how big of a server you need. Microsoft is generally used by gamers for its compatibility and cloud storage as well. If it is a mainstream game this is the choice I would choose even though its more expensive. | One of the great things about mobile games is they are mobile, the customer can play the game wherever they go. Servers can cost anywhere from 20 – 60 dollars a month but with limited computing power its hard to run a full server on one. This would be a great choice depending on the system requirements of the game in use. |
| **Client Side** | Windows and Mac have very similar startup costs. Both platforms are very easy to navigate. If the client isn’t used to using a Mac or Windows it could be quite confusing until they got used to it though. Also Mac is closed source so it may not be as widely distributable. And as a less popular gaming system there might be a lower customer base. | Linux is open source so it can be more widely distributed, and the startup costs would be next to nothing but the biggest problem is that if you aren’t familiar with Linux it has a pretty steep learning curve. It also has a lot less peripheral support for controllers speakers etc. It is also one of the least used gaming platforms so it also has a lower player base. | Windows is the most popular gaming platform because it is partially open source and has great peripheral support but the cost of running a server starts at around $500 a year for a small server. Both Mac and Windows both have the same problem with time. If you have used it before its easy but also has a learning curve if you switch. Most people are either die hard Mac or Windows users. | If the memory allotment and system requirements are very low the program could be tailored to mobile devices so there is a larger customer base and you can rent a server from google for around $20 a month. Most people are familiar with cell phones and tablets so the learning curve wouldn’t be that steep. The only problem would be Apple store and Google Play store which have specific requirements and fees to put your game on their service. |
| **Development Tools** | I think the most popular program to write the game would be Swift or Ecliple which is very widely used among Mac users to write hashnode specifically for Mac operating systems. Swift can also run the very popular IDE called Atom. | The most popular tools on Linux would be Eclipse and Atom. A lot of programs start out being written in Linux because it's easy to write things the way you want. Eclipse is great because it can host multiple languages such as C++ Java etc. | Windows most commonly used is Microsoft visual studio and Eclipse which can host multiple languages as well. If the game is higher end in terms of visuals there are programs such as Unreal engine that can be used on Mac and Windows that help write the code for you in creating objects and landscapes. | For Iphone users they use the same programs as Mac to write their various different applications but are written from a computer and then transferred to that mobile device. For Android things are written for an APK and are mainly written in Java but also from a computer and then transferred. |

## 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 would recommend that the game starts on Windows because of the popularity for gaming. There is a pretty significant cost to get things up and running, but there isn’t as much of a learning curve to use it. There are also multiple IDEs to use for making the game as well as lot of peripheral support. Apple is also another powerful gaming operating platform that could be used but it is very closed meaning they only work best on apple hardware. It could run on a pc but it wouldn’t be optimized and could cause stuttering and crashes.
2. **Operating Systems Architectures**: The great thing about Windows is that users get many different services when gaming or writing a game on windows. When people game on Windows there is things like a GUI or graphic user interface that can be turned on and off as an overlay while playing the game as well as being able to see system resources being used and frame rates. Another great reason to use Windows architecture is being able to test the game on multiple different kinds of systems with different hardware specs to make sure the game runs smoothly. Windows also is compatible with the most high end gaming graphics cards like Nvidia and AMD which help with the amount of rendering that big triple A games have.
3. **Storage Management**: Both Mac and Windows use a storage management system that is called different things like storage sense for Windows. Basically it helps sort and optimize your drives to keep them running at peak efficiency. Also both platforms have many different options for cloud storage for those devices that don’t have a lot of storage on them such as mobile devices. Also being able to choose which pathway you want to store different files in helps to navigate your way into finding things where you want to. For this both PC and Mac are great options. Because Mac is very specific in what they allow on their machines and most programs are optimized to run on them specifically they are slightly faster at storing and pulling from different areas because it helps you choose the best area to store it.
4. **Memory Management**: I would say that both Mac and Windows would be great options for memory because both systems have been tested time and time again for better ways to use memory efficiently. Because Mac is closed source it uses memory slightly more efficiently than Windows does. This game uses many different pictures so using memory efficiently to locate and pull each of the pictures to make the game run smoothly is necessary so both Mac and Windows would work here. Even though Linux has the highest speeds when it comes to memory tests it still isn’t an optimized gaming platform so Windows and Mac use pretty much the same hardware for RAM and its just user preference at that point.
5. **Distributed Systems and Networks**: I have recently started working with an IDE to create my own game called Signs of Winter which can be run on any platform. The newest thing in gaming that many want to see is cross platform play so that a player who is strictly a Mac user can play with someone on Windows, Playstation, Xbox etc. I would say that for outages and connectivity issues with some mobile platforms I would use a server from either Microsoft or Mac to keep the game running smoothly with the internet.
6. **Security**: For security the best is Mac because the systems are so closed and very closely watched by Apple to make sure they stay that way. Whenever you want to add an app or a game to the Apple store there are many requirements that you have to have as well as let Apple have some control over both the software and hardware. Mac also uses a program called Gatekeeper which requires app verification and code signing. They also use X-protect and Malware Removal Tool that also helps to keep third parties from intruding into your system. One problem with apple being known for having the best security software it is also a testing ground for people to try out their malicious software and see if it can break apple’s strict protocols.