

The Gaming 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)4

[**Evaluation**](#_2o15spng8stw)5

[**Recommendations**](#_m8aleynsvzvc)6

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/10/2021 | Peter Bou Saada | Created “Executive Summary”, “Design Constraints”, and “Domain Model” |
| 2.0 | 11/28/2021 | Peter Bou Saada | Created and processed the “Evaluation” section |
| 3.0 | 12/12/2021 | Peter Bou Saada | Created and processed the “Recommendations” section |

**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 Gaming Room is looking to set up Draw It or Lose It as a web-based game. A game like Charades but using stock images to make the guess.

Each image will be drawn on the screen over 30 seconds.

There are multiple teams per game each consisting of multiple players.

If the team playing a round fails to make a correct guess, other teams can attempt to solve the puzzle with a 15-second time limit.

CTS will be responsible to research, setup, and program the environment that The Gaming Room will be using to provide Draw It or Lose It.

## [Design Constraints](#_2et92p0)

Only one instance of the game, a team, or a player can exist at any time, this will be achieved using a unique identifier for each of those. This is a technical constraint.

Each game can have multiple teams. Each team can have multiple players. Teams and players cannot have an instance of game. This is a technical constraint.

The class that runs the game can only exist once. This will be achieved using a Singleton pattern for the class GameService. This is a technical constraint.

The game must be web-based and able to run on all platforms, Mac, Linux, Windows, Android, and iPhone, potentially more. This is a technical constraint.

## [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 is the parent class for Game, Team, and Player. It is there to avoid DRY code (Don’t Repeat Yourself). This class will never be instantiated. It holds common methods and properties between all the entities of our program. This could be an abstract class, but according to the UML diagram it is not since the name of the class is not in Italics.

GameService is the class that will provide us with a lot of the logic behind the flow of the game. It can have many games, and it is a singleton, acting similarly to a driver class.

Game is a child class of Entity. It holds all the properties and methods that Entity holds, and it can have many teams. It represents the actual 4 round game.

Team is a child class of Entity. It holds all the properties and methods that Entity holds, and it can have many players. It represents one team.

Player is a child class of Entity. It holds all the properties and methods that Entity holds. It represents one player.

ProgramDriver is the class that contains the function main(), and it uses the class SingletonTester to test the Singleton GameService.

SingletonTester is the class that provides Singleton testing for GameService to the ProgramDriver class.

**"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:  MacOS allows significant advantages when developing software for MacOS clients.  It also allows the use of a very good command line interface.  Administration on Mac is easier than Windows and Linux.  Disadvantages:  Although it is possible to scale a MacOS server, it is not easily scalable.  It is the most expensive option on this list.  Hardware is limited to what Apple has to offer, and it cannot be easily modified.  Since MacOS is not open source, there might be proprietary software that requires licensing. | Advantages:  Linux is based on Unix, which is open source, and very reliable. Because Linux is open source, there are a lot of software options that we can use to make the server development easy.  Just like MacOS, Linux allows us the use of a very good command line interface.  We can use any hardware we want and install Linux on it.  Disadvantages:  More difficult to use than MacOS or Windows for someone who is not familiar with Linux.  Server setup can be more complicated since Linux does less abstraction than the other Operating Systems. | Advantages:  Windows offers easy server development and administration.  We can use any hardware we want and install Windows on it.  Developing a server on Windows is straightforward.  Disadvantages:  Since Windows is not open source, there might be proprietary software needed, which requires licensing.  Does not provide a command line interface as robust as Linux or MacOS | Advantages:  You can have a non-stationary server.  You can use Mobile Web Server to host a server on mobile devices.  Server does not need to be actively plugged into power. It only needs to be charged occasionally.  Disadvantages:  Mobile devices do not have the hardware capacity to use wired network connections, which are faster than wireless network connections.  Mobile hardware is limited by what the mobile manufacturers are using. Some improvements can be made, but it is not a scalable solution.  Having a physically small server means it would be easier to lose/steal. |
| **Client Side** | Having a MacOS client would work best with a MacOS server, however it is not necessary for the server to be MacOS.  If the client is on MacOS, all users of the client will need to buy a MacOS system, which is by far the most expensive option on this list. | Linux is open source making it the cheapest option on this list, on par with mobile phones, the downside to Linux is the fact that computer graphics support on it is not nearly as good as Windows or Mac OS, however it is still usable. The barrier to entry is not as low as a mobile device. However, it is low enough to allow your average customer to use it. | Windows is by far the most popular operating system available on the market currently. Windows 10 is completely free, only requiring a license for some OS customization options. It can be installed on any computer just like Linux. This puts it at the same barrier of entry as a Linux client. Windows supports DirectX graphics libraries which are objectively superior to any library supported by Linux or Mac OS. | Mobile devices have the lowest barrier to entry out of all these devices. They typically provide an easy-to-use touchscreen, unlike all other Operating Systems on this list.  They are also very mobile due to their size, and the availability of 4G and 5G internet.  Building a client for mobile devices will let the users play while on the go.  This is by far the superior option. It has a low barrier to entry. A mobile device can be bought for around $100 and still support decent graphics. |
| **Development Tools** | A client for a Mac OS system would be written using Objective-C. It can only be compiled on a system running Mac OS. There are ways to compile it on Windows, but they would not provide as good results as compiling on Mac OS. Developers would need their own Mac OS systems to write the client. The IDE required to write Objective-C is called XCode, and it is free on Mac OS not requiring a license. | Building a client for Linux would be best done with C++. Since Linux is open source, there are plenty of open-source IDEs for many different languages. One of the options is visual studio code. QT is another option; however, it is not only an IDE, but a framework for building User Interfaces. This can come in handy. | A Windows client can be written using many different languages and many different IDEs. It is a preference. Java, C#, C++, C are all viable options.  There are a couple free options. Java is completely free using any JDK. C++/C/C# would all require either visual studio or a different IDE like CLion. | Mobile Client development would be easiest with the use of either Android studio for android devices, or XCode for IOS devices. However, it is possible to develop mobile applications using different languages. C++/C#/C are all languages that can be used to make mobile applications. They can be developed on either Windows or Linux.  However, if a web client is to be built. The best option would be to use the Node.JS with a frontend framework like React or Angular. In fact this would work on any platform that supports a web browser. |

## 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**: My recommendation for a server would be Windows. It is by far the easiest platform to start on even though it is a little more costly than a Linux server. It would get Draw It or Lose It to a functioning state much faster than a Linux server. This also opens the opportunity to use a language like C# which is very well supported on windows.
2. **Operating Systems Architectures**: Windows offers services and libraries that make security, Graphical User Interfaces (GUI) programming, API programming, and database development easy. It also provides many web services that allow for easy API publishing.
3. **Storage Management**: Windows provides its own storage management and file system. It is very easy to use and even easier to incorporate into a Java or a C# API. This filesystem allows you to view the amounts of storage taken by extension type. This would allow us to get an easy tally on how much storage is taken by the images being used.
4. **Memory Management**: We can use the windows filesystem as storage for the image files and use a database that holds the name of the image and an absolute path to the file itself. This would require some form of database. On Windows this would be very easy to integrate either using SQL or MongoDB as they are both simple to setup. Application memory management can be controlled easily using pointers in a C++ application. However, this option is not available to us in Java or C#. Both languages do offer garbage collection which does a lot of the memory management for us.
5. **Distributed Systems and Networks**: Since the game is going to require graphical rendering, I recommend setting up AngularJS with WebGL. This combination allows us to use AngularJS to develop platform specific web applications and WebGL to render the image on the screen over the requested. The actual communication between the Clients will go through the RESTful API that is hosted on the windows server. All the dependencies would be standardized on the server. Each client would be responsible to follow the server standard.
6. **Security**: Firewalls need to be used on the server to protect from any potential incoming attacks or connections from illegal sources. User information in the database needs to be hashed. Hashing is a one-way, irreversible encryption. This protects user information in case the database somehow gets leaked. User authorization would use JWT Bearer tokens.