

Draw It Or Lose It

# **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 | 2-23-2023 | Nathan Estrada | Added recommendations |

**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 wishes for their software to have only one instance of their game running at a time. The game can consist of multiple teams, with multiple players assigned to each team. Game and team names must be unique, and players should be able to verify if their desired team name is already in use. In order to produce this software, I will be utilizing singleton and iterator algorithms within the design. The singleton algorithm is a design pattern that only allows one instance of an object to exist at once. The singleton algorithm will be used to ensure only one instance of the game will be running at a time. The iterator algorithm is a design pattern that iterates through each item in a container and accesses their elements. The iterator algorithm will be used to look for existing team names, game names, and game IDs.

## Requirements

The client wants to develop a game that has players on different teams competing against each other on guessing what a slowly rendered image is before the other team can. The client wishes for there to be only one instance of the game running at a time, the option for there to be multiple teams, and multiple players on each team. The client also wishes for each team name, game name, and game ID to be unique.

## [Design Constraints](#_2et92p0)

Because the client wishes for the game to be web-based, this game should be able to be played on both computers and mobile devices, in contrast to the original app only needing to be run on mobile devices. To add, a programming language that offers helpful libraries for web development will be the best choice, as this application is a web game. The client also wishes for the application to only run one instance of the game at a time. The class that will act as main file will need to have functionality to ensure that only one game can be played at a time.

## [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 class GameService contains a list of games, the game ID, player ID, and team ID as private variables, as well as public functions that allow for access and modification of the names and IDs of games, players, and teams. The Game class inherits from both the GameService class as well as the Team class. The Game class represents an instance of a game, and allows the user to name the game, assign it an ID, and add teams to the game. The Team class represents a team of players and inherits from the Player class, allowing players to be added to the team. The Player class represents a player and allows a team to be constructed by several named players. The game and the teams of players are contained in the Entity class, which represents an instance of the game. The game itself is ran in the ProgramDriver class, which acts at the project’s main class, and uses the SingletonTester class which verifies that only a single instance of the game is being ran at once. As described, each of these class represents objects, and certain classes act as object containers for smaller objects, such as the Team class to the Player 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** | One of the major benefits of web hosting using Mac is the higher caliber of security. With Mac, the risk of having unauthorized users accessing data or sensitive information is much smaller than Linux or Windows. However, the downside is that the server would have to run on Mac hardware. If the client’s servers are ran on any other type of hardware other than Mac, they will need to replace it with Mac if they wish to use the Mac operating system. | Due to its open source nature, Linux is the cheapest out of all of these options, and offers a lot of versatility with its multiple distributions. Linux is also compatible with many different types of hardware, and will not require the company to change their hardware for their servers, unless they already have Mac products. Linux is also scalable, and can handle high volumes of users. | Windows is easy to use for hosting web services. While it may not be free like Linux, Windows OS can be ran on multiple different types of hardware. However, Windows does requires a certain standard of hardware (I.e. enough ram and processing speed) unlike Linux. Windows can also run certain applications that Linux and Mac cannot, such as ones that involve MSSQL and ASP. | Using mobile devices to host a server can be a great way for someone to practice hosting servers as well as a cheap alternative to other hardware when running a home server. However, in terms of launching an application or running a business, using a mobile device to host said applications may not be the best idea, as the hardware may not have the processing power required for the scale of the application’s users. |
| **Client Side** | Developing for Mac might be the most complicated out of them all, as Mac software only runs on Mac hardware. However, Mac interfaces are usually the most simple, providing the easiest experience to users. With Mac OS’ tight security, it is important for developers to keep in mind of their software or application not being perceived as a threat by any of the security measures Mac OS provides to allow full access for the products functionality to the user. | Supporting clients that use Linux should be considerably easier than the rest, as Linux offers a lot of versatility and open source benefits. You will not need to worry about developing an application that will only run on hardware that is powerful enough to run Windows, nor hardware that can only run Mac OS. | Windows supports many client side development technologies, such as HTML and CSS. Because Windows can run on a variety of hardware, and Windows also has a relatively simple user-interface, not much expertise would be required for client side development for Windows. | Developing an application for mobile devices is an absolute necessity. The most used operating system is Android OS. Some factors to consider when developing for mobile devices is that mobile devices still contain multiple different operating systems, such as Android OS and Apple OS. It is also important to keep in mind that the user interface for mobile devices is different than that of desktop interfaces, and the application should be easy to navigate with the screen being the dimension of a phone in mind. |
| **Development Tools** | Mac OS comes with a plethora of developer tools already installed in the base product. These tools can be accessed by the default browser Safari. Mac also provides other tools such as Applet Launcher, OpenGL, Thread Viewer, Shark, Alfred, and Homebrew. Mac also has plenty of developer tools outside of exclusives, such as Visual Studio and Git. | Linux has the most freedom when it comes to development due to Linux being open source. Like Windows and Mac, Git can utilize programming languages such as Java and HTML, as well as use multiple different IDEs such as Visual Studio and Eclipse. Developers can also use Windows exclusive development programs in Linux using applications like Wine. | As mentioned before, Windows supports a variety of development tools. These tools include but are not limited to programming languages such as HTML, CSS, Java, C++, and Python. Tools outside of programming languages include Git, GitHub, Visual Studio, and Visual Studio Code. However, these tools do not come with the base OS, and will have to be manually installed | Mobile development is a bit different than desktop development. For mobile development, developers should focus on using languages such as Java, Rust, Kotlin, and Python. These languages include libraries that assist with developing mobile hardware. |

## 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 operating platform that I would recommend for hosting the application is Linux. Linux is not only the cheapest of these options but provides the most flexibility in web hosting services. Additionally, the standard for hardware that is required to run Linux is very low in comparison to Windows and especially Mac.
2. **Operating Systems Architectures**: Linux is an open-source, free operating system. While the quality web hosting services may not be free, Linux can be ran on nearly any type of hardware. There are multiple different distributions of Linux that can be used depending on the company’s needs and budget. Linux also offers the largest variety of web hosting services, such as Bluehost and SiteGround. Linux also runs web applications that utilize PHP and MySQL much better than Windows or any other OS, which gives the development team much more freedom in going about how they build their application.
3. **Storage Management**: The game should come with a simple file explorer feature separate from the operating system’s file explorer. This simplified file explorer will allow users to see the library of images that are used for the game, as well as the option to remove and add images to the library. Different folders can be created that can separate images that can be used for certain game themes or custom play types, as well as images that will be currently used once an instance of the game starts.
4. **Memory Management**: A main memory loading technique that I would recommend implementing is dynamic loading. As the game will likely include a large library of images to avoid the game getting repetitive, it may not be desired to have all of the images loaded into memory at once. With dynamic loading, only the resources that are required for that moment in the game will be loaded. To help manage physical memory, having a portion of the application loaded as virtual memory can come in handy. Virtual memory separates logical memory as viewed by the user from physical memory, which can help prevent the application from running out of physical memory space.
5. **Distributed Systems and Networks**: Knowing that the application will be hosted to many other users devices, and needs to be accessible at any time, ample servers will be required. Servers should be connected to a quality source of power. In the event of a power outage, the servers should rather be connected to a backup source of power, such as a generator, or server rooms for the application should be hosted in multiple locations in the event one of the server rooms loses power. Depending on the scope of the product and how much traction the game gains, it may be a good idea to have servers hosting the application located in multiple areas of the country. For example, if the servers were only located in the west coast, say, California, and someone all the way in Maine wanted to launch the application, they may experience connectivity issues or very slow load times. The number of servers for each region can be determined by how much traffic the game typically experiences in said region.
6. **Security**: A method of security I’d recommend for hosting a web application on Linux is role-based access control (RBAC). RBAC is a security structure where the privilege to access specific processes or data (Such as password information) is assigned to different roles. Role assignment will vary, mostly just between the users and the developers and client. In order to keep these roles secure, each of these roles will be password protected. In order to protect from security attacks, a countermeasure method I’d recommend is cryptography. This way, messages and commands between servers and users' hardware will be decrypted by a key that only the desired recipient has.