

Draw It or Lose It

# **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 | 03/09/2022 | Jason Veno | First Draft |

**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 has a game called Draw It or Lose It. Currently, it’s only an Android app. They wish to develop a web-based version of their game that serves multiple patterns. The game consists of one or multiple teams (each with multiple players) trying to guess a word based on images being rendered from a large library of images in 30 second intervals. One team tries to guess the answer to the puzzle before the image finishes rendering. If they are not able to correctly guess, every other team gets one chance to guess.

A solution to this problem would be to create the program in a way that is compatible with Mac, Linux, Windows, and various mobile devices. By following the UML Class Diagram below, we can ensure that the game meets the specifications of The Gaming Room.

## [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.>

* A game should allow for one or more teams at a time
* Each team should have multiple players
* Games must be unique so that only one instance of a game can exist in memory at a time.
* Team names must be unique so that users can check to see if a team name is taken before choosing one
* The entire program must be compatible with several platforms, including Mac, Linux, Windows, and various mobile devices.

## [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)

**ProgramDriver**: Contains main() method, dependent on (uses) SingletonTester. This is because it obtains a reference to the instance created inside of SingletonTester.

**SingletonTester**: Obtains local reference to singleton instance, used by ProgramDriver.

**Entity**: Base class, holds common attributes and behaviors. Utilizes inheritance. Specifically, Game, Team, and Player all inherit from Entity, because they are all entities. They represent an “is-a” relationship.

**GameService**: Contains list of games, a private instance of GameService (which can be accessed only via the singleton pattern), and several methods that allow us to access information on games, teams, and players, such as how many games there currently are, and the ability to get the name of a team. Has a “0...\*” cardinality with Game, which means there could be zero or multiple games inside of GameService.

**Game**: Contains list of teams, along with the ability to add teams to the game. Has a “0...\*” cardinality with Team, which means there could be zero or multiple teams inside of any game. Also has a constructor that can create a new game when given a unique id and name. There are no mutators/setters, because this information is not meant to be changed once a game is created.

**Team**: Contains list of players, along with the ability to add players to the team. Has a “0...\*” cardinality with Players, which means there could be zero or multiple teams inside of any game. Also has a constructor that can create a new team when given a unique id and name. There are no mutators/setters, because this information is not meant to be changed once a team is created.

**Player**: Holds information about the players, specifically the id and the name. Includes a constructor that can create new players when given an id and a name. There are no mutators/setters, because this information is not meant to be changed once a player is created.

**Summary**: Games are held in GameService, which is a singleton service that ensures that there is only one instance of a game at any given time. Each game can have any number of teams, and each team can have any number of players. Entity is used for simplicity, as it is a base class for Game, Team, and Player, because each of those classes are entities.

**"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** | Second highest cost between Linux and Windows, mainly because it has more functions/features than The Gaming Room needs for their gaming app. Cost also increases as additional servers are used. Runs Apache servers, which makes them simple to use (and learn to use). Excellent graphical user interface. | Linux is open source and therefore free. Uses Apache servers, so it can be run by basic web code like WordPress, making it easy (and inexpensive) to learn how to use. Highly secure with quality speeds. However, there is a learning curve to using it (compared to Mac and Windows), and most of the software has to be installed prior to being used. | Most expensive option, but has numerous features and options, including built-in server-based deployment methods. There are several Windows-based hosts, each with their own advantages, but most of them are costly. Seems to be better equipped for larger-scale projects like The Gaming Room needs. Restrictive user access means high security overall. | Includes Android, iOS, Windows Phone OS, and more. In order for The Gaming Room to deploy their app on other mobile servers, they need to either sideloaded or deeplinked, which are extra steps that take time. This can be done with various supplied installers, some of which are included in packages. Requires the internet at all times. Relatively small monthly costs to host a mobile app, but still more expensive than Linux. |
| **Client Side** | Most clients/users use Safari, Firefox, or Chrome, which many developers will have experience in, making the costs likely low to moderate. | Most clients/users use Firefox, Chrome, or Edge, which is fairly well known to most developers. However, the learning curve of Linux still might cause confusion at first, but is still seen as worth it to many because it is open-source and therefore free. For this reason, development time will be also low to moderate. | Most popular and well known, most clients/users use Edge, Chrome, or Firefox. It is likely that developers will have experience with this, which will likely make converting the app to Windows fairly quick. It is still worth noting that Windows seems to be the most expensive option overall. | Less popular option, which could make finding knowledgeable developers difficult, and might result in the need to train developers before beginning. For this reason, both the cost and time may be high. |
| **Development Tools** | Xcode can be used to make the app compatible with MacOS. It supports several common languages, like C, C++, Java, Python, Swift, etc. | Lost of choices when it comes to IDEs and languages, especially due to it being open-source. This includes popular choices like Visual Studio and Atom, and less popular choices like VSCode and WebStorm. Works with several languages, but primarily utilizes HTML, PHP, JavaScript, and Java. | Also has numerous choices for IDEs and languages, again including Visual Studio, Atom, VSCode, WebStorm, and much more. Languages include HTML, PHP, JavaScript, Java, and much more. | A large factor to consider is that mobile development is not completed on a mobile platform, and requires the use of other machines. A popular option is to use Xcode on Mac, which seems like a wise choice for The Gaming Room. Other options include Visual Studio, VSCode, Atom, and more. Less options with languages, and primarily include lesser known languages like Swift, Objective-C, and JavaScript. |

## 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 Linux OS seems to be the most optimal choice for The Gaming Room, for a variety of reasons. First and foremost, it is easily portable, which allows it to be installed on various types of devices, including mobile, tablet, and computers. This alone makes it a good choice for an app that must work on a multitude of different platforms, but it should also be noted that Linux is open source. This can lead to some confusion given that there’s no standard edition of it, but this will dramatically reduce costs overall.
2. **Operating Systems Architectures**: The Linux OS architecture includes the Kernel, the System Library, the Hardware Layer, the System, and the Shell utility. The Kernel is the core section of the Operating System, and serves as the interface between the hardware and its processes. The System Libraries are special functions that implement the Operating System’s functionality without requiring access rights from the kernel. The Hardware Layer is simply the CPU, the HDD, and the RAM, all of which communicate via the kernel to perform their respective duties. The Shell is the interface between the kernel and the user. Together, all of these Architectures work together to allow the game to run on virtually any platform with general ease.
3. **Storage Management**: The LVM Partitioning Method is commonly used in the Linux OS, for several reasons. It has increased abstraction, flexibility, and control. It also allows for a few helpful features such as allowing you to change the partition size and the number of partitions after installation. When it comes to a file system, the Linux OS often uses ext4, the successor to ext3. This newer version of Linux’s classic file system includes things like metadata, as well as improved performance, reliability, and capacity. The LVM Partitioning Method and ext4 can work together to make an efficient Storage Management system.
4. **Memory Management**: The Linux OS commonly uses Virtual Memory and Demand Paging to create an efficient and reliable Memory Management system. These two features work together to create the illusion of a very large memory, because it utilizes resources in the most efficient way possible. For example, Demand Paging loads pages only as needed, not all of them. This saves precious memory space for other uses, and thus maximizes the memory space provided.
5. **Distributed Systems and Networks**: Draw It or Lose It will need to run across multiple servers, which is a distributed system. This will allow the game to be run on multiple machines at once, regardless of which platform the machine uses. One way to ensure that this process is fluent is with the use of load balancing, which involves spreading client requests evenly among the different servers. Because multiple platforms are being used, communication between the clients (players) and the game should occur via the internet. HTTP allows for there to be multiple sessions of the game at once. Serverless architecture will also support the growth of the game for more platforms and sessions.
6. **Security**: The Linux OS comes with numerous security features that ensure reliable, secure, and efficient usage. For example, each user is walled off from others, and must have password and user ID to use Linux at all. Also, Linux’s file system has discretionary access control, which denies access to all users, unless the owner of the files specifically allows it. Given that Linux is open source, it is highly adaptable and flexible, allowing an experience programmer to limit user’s access rights, restrict the possible flow of malware, and much more. This flexibility allows for maximum security of any program, because the OS can be customized to its specific usages, giving it the least exploitable security flaws possible. This is why reputable tech-based organizations such as *PC World* have stated that Linux is the most secure OS available.