

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## 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 | 09/16/21 | Ken Rodriguez | Added executive summary, design constraints, and an explanation of the UML diagram under the section “Domain Model” |
| 1.1 | 10/03/21 | Ken Rodriguez | Added development requirement evaluation. |
| 1.2 | 10/15/21 | Ken Rodriguez | Filled 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 client, The Gaming Room, has contracted our team to create the circumstances necessary to streamline the development of a web-based version of their gaming app. The Gaming Room has specifically requested that the game support multiple teams containing multiple players, unique game and team names (and a feature to notify users if a game or team name is taken), and that only a single instance of the game can exist in memory at a time. To help accomplish this task, our team will provide The Gaming Room with a software design document and begin developing the application.

## [Design Constraints](#_2et92p0)

* Given that this project will be a web-based version of an existing application, the rules and assets for the game already exist:
  + The application renders images from a library of stock images, this library must be used.
  + A game consists of four rounds, each lasting a single minute.
  + Drawings are slowly rendered over the course of thirty seconds and must use visual clues to solve a puzzle. After the image is rendered, teams have 15 seconds to guess the puzzle’s answer.
* The application must be developed for a web environment; as such, only languages compatible with web browsers should be considered.
* The application must support multiple users through an online network.

## [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 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.**

As pictured in the UML diagram, the ProgramDriver class holds the public method main(), acting as the starting point for our application. The ProgramDriver class uses the SingletonTester class’s public method testSingleton(), a method that will presumably be used to test whether there is a single instance of the GameService class. The GameService class exists to hold information that the Draw It or Lose It game would use throughout the program, containing a list of Game objects that teams would be playing in. The GameService class is built using a singleton design pattern, as such, only one GameService object can exist while the Draw It or Lose It game is played; this ensures that every user accessing the service is using the same list of Games, giving users an opportunity to check if the game name or team name they have chosen is unique, or possibly find games to play with friends. The GameService class utilizes the object-oriented programming principle of encapsulation to ensure that its data is only modified using accessors and mutators (or setters and getters), in turn guaranteeing that the information stored is less likely to be accidentally overwritten or accessed when it shouldn’t be.

The UML diagram also depicts an Entity class; this class serves as the basic framework for Game, Team, and Player classes. The Entity class holds a name and identification number as well as a constructor and accessor and mutator methods to access or modify the Entity object’s name and ID. To utilize another object-oriented programming principle, the Game, Team, and Player classes all inherit most of their aspects from the Entity class; that is, each Game, Team, and Player class will inherit Entity’s constructors, accessors and mutators, and name and ID fields. These classes can then continue to build on the Entity class by adding their own specifically tailored values and value fields. The Player class, for example, uses a constructor and has a name and ID, inheriting these fields from the Entity class. The Team class also inherits these fields while also adding a private list of existing Player objects; a Team can contain zero to many Player objects.

Finally, the Game also inherits Entity’s name and ID fields, but adds a list of existing Team objects. Team objects can be added to a game by searching for existing Team names and a Game can consist of zero to many Teams. Incidentally, the Player, Team, and Game classes utilize polymorphism as each class has its own unique toString() method; while they all are called toString(), they act in different ways and produce different results. As we already know, the GameService class holds each Game object and can be tested using the SingletonTester class’s testSingleton() method. To reiterate, when the Draw It or Lose It game is launched, a single instance of the GameService object is created with the SingletonTester’s testSingleton() method. The single GameService object holds several Game objects that are created on an as-needed basis; these Game objects are full of zero to many Team objects that consist of several-to-no Players; this creates the basic structure of a Draw It or Lose It game environment.

## [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** | Given the “it just works” nature of Mac machines, the selection of available hardware for prospective servers can be very limited. This could be a hindrance in the context of a game, but the hardware used in a Mac machine would have better hardware support out of the box. Additionally, the official macOS Server software offers more features for Mac clients at the relatively low cost of $20 per machine. | Linux servers are generally lightweight, secure, stable, and most importantly, cost effective. Given the operating system’s open-source nature, hosting a server on a Linux system offers a great deal of flexibility and free software with the caveats that technical knowledge is needed to get the most out of a Linux server, and hardware or software support may be lacking. | While Windows systems are less flexible and more expensive than Linux systems and don’t quite share the same hardware ecosystem that macOS machines do, Microsoft Windows is the most common operating system on the market. As a result of its wide adoption, a large variety of server hosting software at a variety of price points is available and developers with necessary experience would be more common. | Mobile devices, as their name would indicate, offer a great deal of hardware portability with lighter devices and operating systems designed to utilize cellular connections; a server running on an Android or iOS device could effectively be set up anywhere and moved with no trouble. On the other hand, these devices usually have weaker hardware and there are few programs designed to host servers on Android or iOS devices. |
| **Client Side** | Mac hardware tends toward the more expensive side of the hardware spectrum and terminal commands can take some practice to learn. This means the monetary cost and the time it takes to learn the software may be slightly higher than comparable Windows software and hardware, but the Mac terminal can offer greater flexibility. | Most Linux operating systems are open source, and Linux software usually follows this philosophy; as such, the software available is free and the terminal commands can be even more flexible than that of a Mac’s. On the other hand, a greater amount of technical expertise is necessary to get the most out of a Linux machine. The learning time investment may be significant when compared that of a Windows or Mac machine. | Given the ubiquity of Windows machines, many applications have been developed for Windows and many developers have likely used a Windows machine. Software will generally cost around that of a Mac, if not a little lower. The required expertise will also be lower than other operating systems as a result. | As stated in the previous section, mobile devices offer a greater deal of mobility compared to other operating systems, low cost, and an even higher market presence than Windows machines; this means that the cost and time investment may tend toward the lower end of the spectrum. On the other hand, mobile devices don’t have a lot of software dedicated to supporting servers, a higher level of expertise may be required to best utilize this software. It’s also worth considering that mobile devices can have varying attributes (physical keyboards, different sizes of touch screens, peripherals) that may make development a more time-consuming task. |
| **Development Tools** | The IDE XCode is included free with Mac machines, supporting the popular macOS and iOS languages Swift and Objective-C. Apple suggests that Swift is the more effective language of the two. | Linux supports a very wide variety of languages, but it itself is written in C; presumably, applications written in C will best interface with the system. With a wide variety of languages, there is also a wide variety of open-source IDEs, such as Eclipse, Microsoft’s Visual Studio Code, and KDevelop. The freedom Linux machines allow is vast, but the documentation available may be sparse. | Microsoft Windows supports a wide variety of programming languages, but of the choices, my favorite is C#. C# was developed by Microsoft to interface with the open-source framework, .NET; the .NET can be used to develop desktop, mobile, and web applications with relative ease. To best work with C#, Microsoft’s open-source Visual Studio Code, an IDE with several extensions dedicated to working with C# that can increase productivity. | <Identify the relevant programming languages and tools (IDEs and other tools) that are used to build this type of software for deploying on Mobile Devices.>  The Android developers have suggested that applications should be developed in Kotlin. To support application development, the official IDE, Android Studio, has been optimized specifically for Kotlin development.  To develop an application for iOS, similarly to a Mac system, the IDE XCode is suggested for development. XCode is included free with Mac machines. Apple again suggests Swift as the most effective language for iOS development. |

## 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**: <Recommend an appropriate operating platform that will allow The Gaming Room to expand Draw It or Lose It to other computing environments.>

Given that the Draw It or Lose It application is written in HTML and will solely interact with devices through a web interface, there is a great deal of flexibility regarding the operating platform the service could function on. I would recommend running the Draw It or Lose It application on **Linux** given the operating’s platform’s stability, cost, and available tools.

1. **Operating Systems Architectures**: <Describe the details of the chosen operating platform architectures.>

Linux systems are based on UNIX operating systems; as such, the Linux structure is composed of three pieces, not dissimilarly to a UNIX system. The first piece being the kernel, where the “meat” of the operating system exists. The kernel maintains important low-level tasks such as process management and memory management. The next piece would be system libraries, which define what programs can and cannot interact with the kernel and subsequently important system processes. Finally, the third piece to a Linux system is the system utilities. System utilities are programs that handle tasks like keeping logs, responding to network signals, and processing login requests.

1. **Storage Management**: <Identify an appropriate storage management system to be used with the recommended operating platform.>

The Draw It or Lose It software will require the use of many images. As such, a hash table algorithm could be used to store these images. Employing a hash table will allow for faster search speeds if a specific image needs to be loaded.

1. **Memory Management**: <Explain how the recommended operating platform uses memory management techniques for the Draw It or Lose It software.>

Linux handles memory in two parts; the first part, the operating system assigns and frees pages and blocks of physical memory. The second part of Linux memory management handles virtual memory that is used by programs and processes. To best interface with the Draw It or Lose It software, I would recommend employing a page sharing method to ensure higher server speeds. The operating system would load each image in the Draw It or Lose It image library to memory when the application is launched, maintaining it in memory until the program is closed. When a user starts a game, the image pages being used in that game would be copied from the pages that were already loaded on the server’s memory. This could be faster than the alternative of loading an image, copying it to the user’s game, then deallocating that memory to repeat the process ad nauseum.

1. **Distributed Systems and Networks**: <Knowing that the client would like Draw It or Lose It to communicate between various platforms, explain how this may be accomplished with distributed software and the network that connects the devices. Consider the dependencies between the components within the distributed systems and networks (connectivity, outages, and so on).>

The Draw It or Lose It program is being developed as a web application; as such, clients will connect to the server hosting the application and interface with the game and other users through a wide-area network (WAN), the internet. To best interface with the host server, I would suggest taking a packet switching approach; the host communicates with the server by exchanging data that is broken into multiple bundles called “packets.” These packets are then given a delivery address (the host to a client’s address or the client to the host’s address, for example) and are sent to be reassembled at the destination. For example, after the host has sent all the packets necessary to play the game such as a login page, a user interface, and a game browser, the client can then use their browser to send packets with their inputs, such as login information, game selections, or game inputs. This approach allows for data to take nonspecific paths through the network which may grant more flexibility if there are connectivity issues or outages.

1. **Security**: <Security is a must-have for the client. Explain how to protect user information on and between various platforms. Consider the user protection and security capabilities of the recommended operating platform.>

Linux systems operate primarily with regards to two major security concerns: making sure that a user is accessing a system that they’re allowed to and making sure that a user has the appropriate permissions to see or modify the files they’re trying to access. To ensure that a user is properly authenticated, a Linux system contains a list of passwords that have been combined with a randomly generated value known as the “salt” value; this value is hidden, discouraging attacks from users without that salt value. When a user tries to log in to the system, the password they have inputted is combined with the hidden salt value and compared to passwords within the Linux directory. If it matches that account’s salted password, the user can log in. This system can be further expanded with libraries that add encryption to the password database, the functionality to allow longer passwords, and so on.

To ensure that users aren’t accessing and modifying files they shouldn’t have access to, Linux systems also employ a permission method in the form of protection masks. These protection masks assign varying levels of access that are complemented to the permission levels assigned to user accounts; for example, a root account may have complete access to fine-tune each aspect of the system, while a guest account would only have enough permission to run a few applications that wouldn’t affect the system in any significant way. Linux systems allow these permissions to be adjusted on a process-to-process basis if users with lesser permissions would need to access applications they otherwise wouldn’t be able to. These systems can be used in tandem to effectively protect both user accounts and the operating system as a whole.