

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 | 5/16/23 | Austin | Completion of Executive Summary, Requirements,  Design Constraints, and Domain Model. |
| 2.0 | 6/15/23 | Austin | Completion of Recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

The main goal of The Gaming Room’s Draw It or Lose game is to be accessible as a web-based game across multiple platforms. The game is currently only available on Android so other users using iOS for example are not able to play, a web-based app would resolve this issue. The game steadily renders an image while team members try and guess what the image is (a phrase, title, or thing) before the time expires. If the original team is unable to guess correctly, opposing teams can submit one guess within a 15 second time limit. While doing so we will create a streamline process for development and choose the best environment for the web-based game.

## Requirements

There are a few requirements for the Draw it or Lose It game. The first requirement is that a game must have the ability to have more than one team involved in a game. For the game to work best, there is an opponent each team is playing against. Another requirement is that each team will have multiple players assigned to a team. While creating these games and teams, each team and game has a name. Another requirement is that no game or team within the game is allowed to share names, all names must be unique. The last requirement is that only one instance of the game can exist in memory at any given time.

## [Design Constraints](#_2et92p0)

One of the bigger constraints with user a web-based game is the need for an internet connection. With a mobile app there may be options to play offline but with a web-based game that is certainly not an option. Another concern and constraint are the ability to make the web-game accessible on all platforms. This means someone can play on a mobile device, laptop, desktop, tablet and not have any issues with the scale of use of the web-app but also with the different OS each device uses. A similar issue is browser compatibility. Making sure the user can access the game on safari, chrome, Firefox, or whatever it may be, if that is the goal of course. With the use of browsers and the web in general come the security risks. Another constraint is making sure the website is scalable and allows a large base of players to access and play the game, but also making sure that the game is secure for both the company and the users. There could be security threats to The Gaming Room from hackers and unauthorized access, but also cheating and unauthorized data collection from the users playing the game.

## [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 ProgramDriver and SingletonTester classes are related by association as the ProgramDriver class uses the SingletonTester. The ProgramDriver class contains the main which runs the rests of the program. The Entity class is the base class for the Game, Team, and Player classes. All three of these classes inherited from the Entity class. The Game, Team, and Player classes all use the id and name attributes as well as some of the methods. The Player, Team, and Game classes are also all associated. The Player and Team class share a none-to-many relationship as does the Team to Game classes. This means that there can be none to many players on a team and none to many teams in a game. The last relationship is between the Game class and GameServices. This is again a one-to-many relationship. GameServices is associated with the Game class and provides several methods to control the created list of Games in the class. Teams are created from lists of players, Games are created using list of teams, and the GameServices is created based on list of Games. In doing so, we can make sure we meet the Gaming Room’s requirements of unique game names and team names. This will also allow us to add multiple players to a team and multiple teams to a game.

**"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.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac is known for its ease-of-use good performance. Apple’s Mac is a very user-friendly development tool that doesn’t require a large learning curve. Mac is very easy with other Apple OS such as iOS, but using it cross platforms can get complex when natively hosting a app. With a web-based game this isn’t as big of a concern as Mac is highly compatible with web-based tech and frameworks. Mac is expensive compared to Windows and Linux and is often more expensive as there is only certain compatible hardware. Mac is not as popular for server-side applications but can be done. | Linux has several deployment options such as dedicated servers and cloud platforms. The openness of Linux and the numerous distributions paired with its highly customizable and flexible nature makes it a great tool for server-side development. Linux is stable, secure, and vary scalable all while maintaining a very low cost, if incurring any one at all. Linux is most complex to learn and takes some practice so that is one of the major drawbacks from someone unfamiliar with Linux. | Windows is a very popular development system used heavily by both personal and business environments. Like Apple, Windows in Microsoft’s OS. Windows, like apple, can have licensing costs attached. Windows is very user friendly and has a vast range of software support from commercial to gaming. Hardware is much more customizable due to the ability to interchange parts. Windows has a Windows Server which was for sever deployment. Windows is still not as popular as Linux for hosting web apps but more popular than Mac. | OS such as Android and Apple are almost never used for Server-Side development. These devices are small and mobile which makes them great for certain tasks, but less robust compared to computers and Mac, Windows, and Linux. There user experiences are simple and less customizable. Licensing is generally always free with the purchase of a device. Licensing can’t be switch from one device to another, such as Android on an Apple device. |
| **Client Side** | When creating a client side on Mac it would be important to have browser compatibility with the browser used my Max users – Safari and google are the most popular. It would also be important the users are able to see a uniform game across iPads, iPhones, Mac laptops and other devices. This just makes sure that the user experience is equal across all devices. The client side would need to be done using common web technologies such as HTML, CSS, and JavaScript to ensure the ability to access it on other operating systems. The time to do it this way would be consistent across all systems due to the similar style of development and common practices. | If the team is inexperienced with Linux, this will be a big learning curve. One that will cost both a lot of time and money. Windows and Mac are easier to pick up and learn. Linux has a variety of cross platform distribution tools that would allow easy use across platforms. Linux has a large open-source community that can be used in development. Development would still be done in HTML, CSS, and JavaScript so the development time in terms of real coding would be like other systems, assuming there was no friction in learning how to use Linux. | Windows is a popular development tool that, if someone is not familiar with, will not be too time consuming and costly to learn. Using the same methods as before, developing in JS, HTML, and CSS, we can ensure the client side can be run on multiple systems and browsers. Testing on Windows using different versions on Windows to test compatibility, functionality, and performance. Creating the client side on windows for a web app allows access on other systems and mobile devices. | Mobile Devices rely on touch which is different from most other systems which rely on mostly keyboard and mouse inputs. This being said, a mobile device client side must have touch controls and there are no keyboard inputs. This requires a different approach that can become more time consuming and costly to learn. The overall development on a mobile device is much harder as it a much smaller device and there is much less control over the device. It is hard to track and edit files and well and test existing ones. This can be done using the same HTML, CSS, and JS and mentioned before but certainly isn’t popular and certainly isn’t conventional. |
| **Development Tools** | Languages that will be used are HTML, CSS, and JS. While suing Mac there are some common frameworks such as Unity that can be used to develop the game on numerous platforms, web included. A Mac popular Mac IDE that can be used is Visual Studio Code. The is a lot of customizable aspects that are helpful for web development. A version control that could be used on any system is Git. This would allow us the track, edit, and update code as we develop the game. The game also needs to be tested and some popular testing frameworks could be either Jest or Cypress. It looks like Cypress may be the favored one. There would be minor costs associated with Mac development. Some may come from upgraded version of an IDE or a Git hosting service like GitHub. | The languages that would make the most sense with Linux are still HTML, CSS, and JS. This is the standard for a web-based app and the game fits right into that. When creating the front and backend for the game something like node.js would be used on the server side and framework like react of angular could be used on the frontend and creating the client-side interface. There would also need to be some sort of compatible database as well. Again, as with Mac, Visual Studio Code is a good option for an IDE. Like Mac, version controls like Git would also be helpful in writing and maintaining code across the team of developers | Similarly, the languages that would make the most sense with Windows are still HTML, CSS, and JS. Mac and windows have a more user-friendly interface opposed to Linux, but the same tools and frameworks can be used. Windows will also use a front and backend framework such as Node.js or .Net in the back and React or Angular in the front. There is still a need for a database and something like Microsoft SQL could be used with Windows. It may be more efficient to have two teams working on both the server and client side. Still, VSC or VS will be great for developing on Windows and Git is still most likely the preferred version control for a lot of developers. | The languages that would make the most sense when developing a cross platform web-app on mobile are still HTML, CSS, and JS. Anything web generally utilized this stack and mobile devices are no different. The cross platform mobile apps can be different when it comes to frameworks. The others used a different front and back-end framework, mobile however can use something like Apache Cordova or React Native to construct both the front and back end. The most popular mobile devices are Apple and Android and so we can say that Android Studio or Xcode would be viable development tools. VSC would still be an option, and the preferred method unless the entire team was using the same mobile OS. This could create the need for multiple teams. |

## 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 The Gaming Room is to use Linux in their expansion of the Draw It or Lose it game. Linux is well known for its stability and security which makes it a commonly used server for hosting web applications. While other platforms, such as Mac and Windows, are limited in their distributions, Linux is open source and has several distribution tools to choose from. Another benefit of being open source is the vast number of available resources and documentation from the user base and community. Choosing the right platform comes down to the task, but overall Linux’s lightweight and efficient nature would make it a great choice for web-based server work.

1. **Operating Systems Architectures**:

The Linux platform is comprised of a few key main elements which include the Kernel, System Library, Hardware, as well as Shell functions and applications. Linux uses a monolithic Kernel that can be used to control memory management, drivers, file system and much more. A system like Mac uses the XNU Kernel which is a hybrid, using both aspects from the monolithic but also the microkernel. Both share some similarities to UNIX such as the shell and file permissions. Linux is open source which allows users to freely access and modify the source code. Users of Linux can choose a distribution that allows access and modification of system libraries and other aspects that are unviable on other systems.

This is the opposite for Mac and Windows. These are proprietary systems that keep source code hidden from the public. With customization in mind, Linux offers several GUIs and desktop environments whereas the other systems come pre-installed with the GUIs and software needed to run the set system. Linux is also much more customizable when it comes to hardware components. If they are compatible, you can really use any combination you would like for parts. This creates another layer of flexibility when it comes to device drivers and managing communication between hardware and the system.

**Storage Management**:

There are many storage management systems for Linux. Some of the more popular ones I have come across are LVM (Logical Volume Manager), RAID (Redundant Array of Independent Disk), XFS (X File System), and Ext4 (Fourth Extended File System). For a lot of Linux distributions, Ext4 is the default file system. LVM allows a user to manage the storage volumes such as the HDD and SDD. This gives the user more control and allows for resizing and grouping. RAID is a storage management tool that follows the practice of combining multiple physical hard disks into one single unit. This improves performance and storage capacities.

The storage management system that I would recommend for the Gaming Room while using Linux would be XFS. XFS is known for high performance and is well suited for large-scale systems. This web game could become extremely large - tracking teams, users, and photos so extra performance and storage are essential. XFS offers high speed access, which is needed when playing a real-time game. Drawing current data or accessing the photo bank needs to be efficient. XFS is known for its ability to manage large workloads and is commonly used in enterprise and server environments, so it fits right into the Gaming Room’s needs.

1. **Memory Management**:

The three most efficient and important memory management techniques on Linux consists of virtual memory, paging, and caching. Virtual memory is a foundational method in managing memory on the Linux platform. Virtual memory gives the ability to allocate a virtual memory address to each process. This virtual address creates isolation between processes which ultimately allows each process to operate as though they are access to the entire memory capacity. This has other benefits such as efficient sharing and memory protection as well. Another technique that can be layered on top of virtual memory is paging.

Paging breaks up data when the physical memory is low and releases unused pages to the swap disk space to free up needed physical space for another process. A page is essentially a chunk of data that can be stored using virtual memory. Paging is a way to break up and store large amounts of data so that large amounts of memory aren’t used when only a small part of the process is needed. Paging is essential and a building block that leads to virtual memory.

The third technique for memory management is caching. Linux and other OS use page cache. This cache holds frequently accessed data in a quickly accessed part of memory. By keeping it in an easy-to-get place, it makes the retrieval more efficient and much faster, eliminating the longer time needed for searching the disk. This data is changed, removed, and added to bases on different processes being run, keeping the most essential parts of the memory easy to access.

**Distributed Systems and Networks**:

Using distributed systems will allow the team to break down the game into its smaller subsets. Instead of one process running the front-end interface, database, servers, breaking them up into separate aspects that communicate through standard protocols, such as IP or HTTP, allows for modular development. Looking at the game there are three main parts. The servers control the game’s logic, running the timers, loading the images, and controlling matches and player communication - all the behind-the-scenes actions. While having a single server may be okay for smaller projects, a game of this scale is going to need a group of servers. Looking at almost every other poplar game, there are a series of different servers serving different locations, either globally or around the country. The bigger the game is, the more servers there are and the closer together they are. By doing this, users can connect and play off the closest server, decreasing latency and creating more efficient game play. The servers will communicate amongst each other sharing information as needed. In the case that one of the servers goes down, another server can absorb that load to keep from a total failure.

The servers will also communicate with another major part of the game, the web interface. The web interface will be what the players use to communicate with the server. This will be through clicking buttons, typing in the chat, or any other incorporated method. The web interface will interact with the game servers ApIs using various HTTP methods such as GET, POST, or PUT. When the web application needs to draw information such as user details, scores and game details, or another information, it will create a request to the server to pull information from the database, the third major component of the game.

The database will store and organize all the essential information of the game so it can be quicky queried and respond to request from the web server. The web server will create a request to the game servers which act as the middleman. The server then stores or retrieves the needed data and sends a response back to the web interface. The server is needed to mediate the communication between the interface and the data storage. All three aspects of the system rely on strong communication through a network protocol. To keep the speed of the game running at the levels needed to perform, the requests need to be fast and efficient. This can become slower if there is not enough bandwidth which will hurt performance. This can be prevented using certain sequences such as data compression which helps to avoid these issues. Other fixes may just balance the servers so that one doesn’t get overused.

Overuse of game servers can lead to outages as well. This can be caused by a few things such as performance degradation, exhaustion, or eventually denial. Degradation is where the server starts to see slower performance over time. This can result in slower request and response times which will lead to delays on the user’s end. Exhaustion comes from overuse of the servers’ resources. This can lead to a spotter server, one which drops requests and causes disruptions. A server denial is an extreme case where the server simply refuses to handle additional requests. Its current load becomes overwhelming which can lead to outages and a load of other issues. This can be prevented using a few methods. Monitoring the existing servers and having a plan for scalability is the most important. Making sure the existing servers are running smoothly can prevent overuse and later consequences. When servers are starting to get more use, expanding horizontally, and adding additional server to cover the load is often an inexpensive process compared to vertical expansion. The last preventive method is for different types of load testing. Stress testing the servers will show inefficiencies and problem areas that can be fixed ahead of time.

1. **Security**:

There are several different methods that could be used to deal with the Gaming Room’s security, the most beneficial methods would be preventive ones. This includes different techniques and methods such as encryption, access control, network security, and data backups. It’s important that the Gaming Room’s application takes steps to actively prevent security breaches opposed to just reacting to them.

Encryption is a good way to protect sensitive information. This can be both user information and enterprise information. Encryption scrambles information being passed from one point to another so that if it is intercepted or the source is compromised, the data can’t easily be accessed and used. In a way this is double preventive. The first part makes sure that someone cannot see what is being transmitted and the second part ensures that if there is a breach, the unauthorized user can’t read and access the data. Encryption isn’t a one stop fix to security, however. Encryption can be cracked, but at least it gives some time to then react and hopefully fix the issues before it causes too much damage.

Another technique to layer on top of encryption and another common practice is access control. Simply put, this is the authentication of a user before the user can access sensitive information. Commonly this is done through passwords, two factor authentication, and other methods. This practice makes it so only the owner of the user information can access their information.

The last preventive step is on the network. The network allows for communication between the server, interface, and database so it is important to secure these requests. Many organizations install firewalls, protecting users from suspicious websites that could cause security issues. Firewalls could be run on the Linux system to guard against breaches and allow certain information to flow in and out. Another common practice is to use VPNs. These Virtual Private Networks help create a more secure communication between servers, platforms, and networks. This will be a beneficial tool for the Gaming Room as they will have communication across all three.

Overall monitoring processes are greatly beneficial for the system and security. Being aware of any unnatural occurrences and possible threats can eliminate them before they become dangerous and is something that should be taken seriously and into account on all projects involving sensitive information. It is not just the Gaming Room’s information they must protect, but all the information the users have provided and trust the Gaming Room to keep secure.