

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 5**

[**Domain Model 5**](#_Toc115077324)

[**Evaluation 7**](#_Toc115077325)

[**Recommendations 11**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 07/18/25 | Shawn Stover | Added Executive Summary, Requirements, Design Constraints, added explanations to Domain Model |
| 2.0 | 08/02/25 | Shawn Stover | Evaluation Section completed |
| 3.0 | 08/16/25 | Shawn Stover | Recommendations Section Completed |

**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)

There are a few things that need to be addressed in order for the game development of the game for The Gaming Room. Teams will make guesses at drawings as they are being rendered. Instead of teams drawing and allowing their teammates to make guesses, the images will be slowly rendered from a library of images within the program. Games should consist of four one-minute rounds. The drawing will be fully rendered by the thirty second mark in each round. If the team does not guess the answer within the time limit, the other teams will have a chance to guess once each within a fifteen second timer.

The game should have the ability to have one or more teams involved and each team will have multiple players assigned to it. Team and game names must be unique; there should be systems in place to check that the name is in use when it is being created. One instance of the overall game can be in memory at a time. This would be accomplished by ensuring that there are unique identifiers for each game, team, and player. The game also must exist in a web-based distributed environment.

## Requirements

The team is unsure how to set up the environment and that is where I come in to ensure that the environment and the game meet their expectations. The game must run in a web-based environment. The software requirements are as follows: the application should render images slowly over thirty seconds from a large library of stock drawings. Games consist of four one-minute rounds and the guessing team needs to guess before the time runs out. If unsuccessful, the other teams each get one guess within a fifteen second time limit. Hardware requirements will be added at a later date.

## [Design Constraints](#_2et92p0)

There are a few key design constraints that must be kept in mind before development begins and while development proceeds. Since this is going to be a web hosted game, the game must be able to run on multiple devices and platforms including desktops mobile devices and tablets. This is going to require responsive design decisions so the game displays and runs properly on a multitude of devices as well as ensuring that technologies are used that will work across multiple platforms. These technologies include HTML, CSS, JavaScript, and a platform neutral backend such as Java. This would ensure that the game would run well across a multitude of platforms and devices and allow the most players to enjoy the game.

Another constraint that is important here is session management. This means that there is only one instance of the game running at any one time. This necessitates the use of singleton patterns, or a way of allowing a single instance to exist and allow global access to it within the application./ This will make development easier and in the end more secure for users. This helps control the ways things are handled in memory and ensures that things are running the way they should.

Name uniqueness is another constraint that needs to be handled. Names need to be unique across teams, games, and players. This ensures that the code does not get confused when differentiating between one entity and another. Checking these new entries against existing records to ensure uniqueness allows the program to run smoother across all devices connected, the game will not suffer from difficulties by getting confused between entities and this will make the game run smoother overall.

While we are looking at ensuring the game runs across many platforms and on many devices, scalability and performance are also paramount. The larger a single game becomes with more teams and more players, the more important scalability is. Regardless of the number of entities involved the game should render drawing efficiently and handle game state transitions well to ensure success of the product.

The game data should also be secured and handled by role-based access sanitizing inputs and ensuring that session isolation is key. This will ensure that the game is as anti-cheating as our players would expect it to be. The better we are at deterring and punishing cheaters the larger our community will grow as we show them that we believe in fair play and ensuring that games are secure and not manipulated in any way.

Across all of these constraints lies the issue that implementing these features will impact development time and affect the overall bottom line of the product. However, spending the extra development time now to implement all of these features will increase the games word of mouth and ensure that our player base grows as strong as it can be. This increases the success of the game, the company’s bottom line, and ensures that players are happy with our product and will play it for a long 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 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.**

With the given UML diagram there are some key concepts we can explain. The Entity class is used as an abstract base class that handles all uniquely identifiable objects such as those that share the id and name attributes. GameService is the singleton class that manages game creation, retrieval, and ID generation for the game. Game repres3ents an existing game session containing multiple Team objects, one for each Team in the game. The Team class represents a group of players within a game, encapsulating a list of Player objects, one for each Player. Player represents an individual participating in the game. ProgramDriver is the entry point for the application, this triggers game functionality through SingletonTester. SingletonTester exists as a utility class to verify the singleton behavior of the GameService class.

There are some class associations we need to discuss across the application. There is inheritance existing because Game, Team and Player all inherit from the Entity class, this ensures that things such as is, name, and behaviors like toString() can be reused across the program without issues. There are some aggregation relationships to point out as well: These are:

GameService > Game (0…\*)

Game > Team (0…\*)

Teram > Player (0…\*)

These aggregation relationships show a cascading hierarchy where one GameService can house and subsequently manage multiple games, each game handles multiple teams, and each team handles multiple players. This ensures everything stays orderly and makes sense for the program to use and keep track of.

There is a single usage dependency as well, ProgramDriver uses SingletonTester, this ensures that there is logic in place to ensure that the singleton is validated before use without directing modifying the business logic of the program itself.

There are a few Object-Oriented principles at play as well. Encapsulation is being used so that all classes keep their internal lists and identifiers private while being able to be accessed by the public accessors. This ensures that all data is stored in capsules and accessed when needed in an orderly fashion. There is inheritance throughout the entity class, it provides an interface that is common and allows for the identifying of objects, this helps limit code duplication. Polymorphism exists in the toString() method, it is implemented in a unique way by each subclass, allowing output overrides that are meaningful for each related class. There is a Singleton pattern in place as GameService uses getInstance() to restrict instancing ensuring a single memory area for game management.

Alignment with software requirements is great throughout the diagram. The singleton design of GameService and the related tester ensures that only one manager exists at a time in memory. Unique team and game names make it easy for the program to identify and subsequently handle each Entity efficiently due to uniqueness checks in name attributes. The team-player hierarchy and aggregation ensure that Game, Team, and Player mirror the required gameplay flow and ensure clean scalability across games. ID Generation is handled by the GameService class handling identifier creation through nextGameId, nextTeamId, and nextPlayerId reinforcing consistency and avoiding program crashes

## [Evaluation](#_2o15spng8stw)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac will have a rough time as a server, MacOS Server was depreciated in 2022 and they stopped selling servers systems before then. With these issues in mind, while Macs can be used for continuous integration / continuous deployment practices and ease the process of integration with ode merges and deployment through ensuring that code is updated and deployment is streamlined, it would not work well as a standalone server. MacOS comes bundled with the system and will not incur an additional fee. | Linux would be an ideal platform to host scalable web apps using Apache, NGINX, Docker or Kubernetes, it also fully supports any cloud-based architecture and orchestrates the use of containers easily. This allows Linux to host servers within the landscape of these specific architectures and would be able to be used for server hosting. Most Linux distributions for Ubuntu and Devian are free. However, Enterprise support through Red Hat or SUSE ranges from $350 to $2,700 / year depending on the size of the enterprise. | Windows has strong support for backend development and running a dedicated backend through the use of strong IIS support, .Net Core and Azure. This makes it ideal for enterprise scale deployments that are looking to scale to a large number of endpoints and concurrent clients. Licensing costs are somewhat steep with Windows Server Standard being $1,176 and datacenter editions being $6,771 plus relevant Client Access Licenses. | The ability to run a dedicated server through a mobile device is virtually non-existent. Mobile devices rely on cloud or edge servers to receive data and receive server-side instructions as a client. There are some licensing costs associated with this although the Android SDK is free, Apple Developer Program will run a corporation $99 / year. |
| **Client Side** | Mac fully supports HTML5 and will work well when used to run the game client through the web. Further, Mac is excellent for testing iOS compatibility, which would be great for an iPhone version of the game. It would be best to have a dedicated team of Mac specific developers due to the nature of developing for MacOS and the systems intricacies. This would also potentially add some time to the development process. There are higher hardware costs associated with Mac, the need to use Apple’s developer program which is $99/year. This is essential for iOS building and testing. | Linux fully supports modern browsers like Chrome and Firefox. Any responsive HTML5 apps run seamlessly within these supported browsers. There are a few things to keep in mind with Linux though. This OS requires Linux-savvy developers for the program to be able to run on Linux well, but there is a strong open-source community that can help developers become more versed in Linux and developing for Linux. Linux as an OS is excellent for backend and DevOps teams that are looking for a great platform to host a server application. | Windows has a good selection of browsers for testing the product. Edge, Chrome and Firefox all support responsive HTML5 and have broad enterprise adoptions due to the popularity of Windows based machines. With this being said, there is less of a need for Windows specific developers and developers can instead focus on ensuring that the server runs well through a Windows backend and is scaling and responding corrects to the different clients connecting. This has less focus on specific strong developers and more of a focus on ensuring stability and delivering a quality product to customers. This means lower onboarding times, and a strong support ecosystem despite the higher licensing costs. | For the clients, HTML5 runs natively within mobile browsers, however, responsive design is imperative to ensure that mobile users get a good experience. It is important that there is mobile device testing being done during the development process to ensure that the program will run and work on mobile devices correctly for a good experience for the consumer. This requires the use of cross-platform frameworks such as Flutter and React Native to make the mobile experience the best it can be. Mobile development also has a moderate learning curve with a unified codebase allowing for reduction in duplicate code. |
| **Development Tools** | There are a few development tools available for Mac. A Mac application could be built in Swift, Objective-C or JavaScript. There are also as few IDEs available to Mac. VSCode, Xcode, which is more for iOS development, and IntelliJ. | Linux has access to a few languages and IDEs to allow for development within a Linux system. The languages available for program to run in are Java, Python node.js and PHP. IDEs available are VSCode, IntelliJ IDEA, Eclipse and NetBeans. This gives developers options for the IDE they like the best without being pigeon-holed into a one size fits all solution. | Windows has a few languages that programs can run in. These are C#, JavaScript, and Python. These allow robust programs that deliver a quality product. IDEs available for Windows are VSCode, Visual Studio and JetBrains Rider. This gives developers great options for deciding their IDE of choice that still allows them to code effectively for a Windows program. | Mobile devices offer a robust selection of languages with Java, Kolton for Android, iOS for iPhones, Dart with Flutter and JavaScript specifically React Native. IDEs that are available catering towards mobile development are Android Studio, Xcode and VSCode. A Mobile development team is going to require a dedicated team that understand that intricacies and challenges of developing for a mobile platform. |

## 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**: Linux should be used for the server. As stated in the server-side section of the evaluations, Linux has some distributions which are free, and they also offer sliding scale cost Enterprise support for when the game grows. Linux is highly scalable and customizable for the game’s needs. Further, it offers excellent performance for gaming backends, which is exactly what The Gaming Room needs as Draw It or Lose it gains popularity. Lastly, it has robust security features and frequent updates to improve security which allows the user accounts and overall game security to be intact and strengthen with each update. Also, The Gaming Room should look into Cloud Hosting such as on AWS, Azure Virtual Machines or Google Cloud’s own Compute Engine. This would help with flexibility, scalability and enable a global reach for the game allowing it to grow as quickly as possible.
2. **Operating Systems Architectures**: Linux’s architecture allows it too is monolithic in the sense that it allows all OS services to run directly within the kernel space. This allows for the system to have high performance due to being able to make direct service calls. Linuz also offers a very modular design allowing customizations on the server side for performance and stability that will preserve and protect the game sessions allowing security and fair play. Linux also provides the handling of concurrent game sessions and user interactions; this combined with the flexibility and scalability Linux provides for the server side allows Linux as an operating system to be the ideal platform to not only handle data but transfer data back and forth between players and the server.
3. **Storage Management**: Linux also has the ability to support EXT4 and RAID storage for system redundancy to allow the server to be stable even when sections of the RAID may go down. RAID 10 allows the server to mirror and stripe various drives to improve stability and reliability. Also, EXT4 as the file system allows for large file support, journaling to protect against crashes, and performance features such as delayed allocation that allow for superb storage on the backend and fast, reliable servicing for the clients to keep the game running smooth and being responsive to players. Fast read/write speeds allow for real time multiplayer data, essential for a game like Draw It or Lose It.
4. **Memory Management**: Linux is ideal for memory management as it allows for the use of Virtual Memory, allowing for extended physical memory to allocate larger game sessions with more players or simply more game sessions in total. Linux can also use swapping, allowing pages to be swapped onto disks to free up RAM to allocate for more effective running and management of the game sessions. Linux can also cache data, allowing it to access frequently used files, such as those related to player data, scores and the images for the game quicker and make the game run smoother. Within the games code itself, The Gaming Room can also employ garbage collection and object pooling to allow the game code to execute faster, more reliability and improve the overall service of the game for users.
5. **Distributed Systems and Networks**: Draw It or Lose It can employ microservice based design which allows game logic, user management, and scoring into separate services for smoother servicing. Thes services can then be deployed across containers using Docker and then servicing orchestrated using Kubernetes, allowing efficient and consistent services across platforms. RESTful APIs can be used for most game logic and communication, this will allow the most clients on the most different systems easily be able to interface with the game and allow the user base, player base and concurrently games to grow over time. WebSockets are going to be key as well to allow for real-time gameplay updates across various platforms and systems, keeping the game running smoothly regardless of which client a player is using to connect. Server networking can be distributed effectively using Load Balancers for effective distribution of traffic. Failover mechanisms are also going to be important, allowing outages to be managed as best as possible, this way the game service will last as long as it possibly can before being lost completely due to network outages. Lastly, latency and throughput can be monitored by software such as Prometheus and Grafana to ensure that the game is being serviced quickly and efficiently so the players are enjoying themselves with as little lag and disruption as possible while playing.

1. **Security**: Security is an essential part of running a successful service. TLS and SSL should be used for all data in transit, sensitive data such as user account information and anything needed strictly by the server should be encrypted using AES-256. Role Based Access Control should be used for administrative needs for users. OAuth 2.0 should be used for secure user sessions that will protect the security and integrity of the game. Regular maintenance updates should be released for not only game updates but patching vulnerabilities, also Linux firewall and SELinux and AppArmor can be used by the server to ensure access control is secure for further game integrity. For distributed security, secure API endpoints including rate limiting and input validation which will ensure that VPNs or privates subnets should be used to protect and validate internal server communications to ensure packets can’t be intercepted and modified.