

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## 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 | 09/17/23 | Ivette Cerpa | Initial executive summary, requirements, design constraints, and domain model added |
| 1.1 | 10/01/23 | Ivette Cerpa | Evaluation section completed |
| 1.2 | 10/15/23 | Ivette Cerpa | Recommendation 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)

The Gaming Room has hired Creative Technology Solutions (CTS) to develop a web-based version of their game, Draw It or Lose It. Currently available only as an Android app, Draw It or Lose It is a competitive drawing game where teams compete to guess what image is being rendered from a library of stock images.

The Gaming Room has specified that the game application be multi-team capable, have multiple players per team, allow unique game and team names, and that a single instance of the game exists in memory.

CTS will prepare a comprehensive software design document addressing these requirements and begin developing the game application. The hardware requirements will be determined later as a result of the software application decisions.

## Requirements

* **Multi-team capability**: A game should have the ability to involve one or more teams.
* **Multiple players per team**: Each team should be able to accommodate multiple players assigned to it.
* **Unique game and team names**: Game and team names must be unique. This allows players to check if a team name is already in use.
* **Single instance of game in memory**: Only one instance of the game can exist in memory at any given time. This can be achieved by creating unique identifiers for each instance of the game, team, or player.

## [Design Constraints](#_2et92p0)

We are developing Draw It or Lose It into a web-based application that must be compatible across various platforms. Given its accessibility from various devices such as computers, phones, and tablets, the game must operate seamlessly on all these platforms. This requirement will directly influence the design of the user interface and game controls.

Draw It or Lose It will accommodate multiple teams with multiple players and must be designed to handle these simultaneous interactions consistently. A robust server infrastructure and efficient synchronization mechanisms will be required to ensure a smooth and fair gaming experience for all players. Additionally, having unique game and team names, as well as only having one instance of the game in memory should help with the game’s stability and performance.

Prioritizing the security of game data is a necessary constraint. We must ensure the game's integrity and players' privacy by implementing robust security measures, including encryption and security protocols.

For a web-based application, optimization and performance are critical. We must ensure the game runs efficiently on the user's preferred web browser and device. To accomplish this involves reducing load times, fine-tuning graphics rendering, and guaranteeing smooth gameplay, which will significantly improve the user's gaming experience. Additionally, as the game will utilize a library of images for rendering, this aspect must also be factored into the design and optimization process. It is crucial to employ effective image-loading strategies to ensure the swift loading of images without hindering the game's overall performance.

Draw It or Lose It will host multiple players simultaneously and render images; network latency becomes a crucial factor influencing the gaming experience. We must implement effective latency management strategies into the game's design.

## [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.**

The ProgramDriver class uses the method from the SingletonTester class to verify the application of the Singleton pattern.

The GameService class shows zero-to-many multiplicities with the Game, Team, and Player classes. In our design, a single instance of GameService will exist and manage an arbitrary number of Game instances. Each Game will manage an arbitrary number of Team instances, and each Team will manage an arbitrary number of Player instances. This structure allows for a single instance of GameService to be maintained in memory, while the number of instances for Game, Team, and Player can fluctuate based on requirements.

The Entity class demonstrates the concept of inheritance by serving as a superclass for the Game, Team, and Player subclasses. Shared attributes among these classes are contained within the Entity class and inherited by the subclasses.

Polymorphism is present through the overloaded getGame method in the GameService class and the overloaded constructor in the Entity class.

The UML diagram shows the presence of encapsulation through the use of public (+) and private (-) access modifiers.

The overall game is divided into separate components to depict the concept of abstraction. Each component abstracts unnecessary details by encapsulating pertinent properties and behaviors. The Game, Team, and Player classes encapsulate only the relevant information to each of them. The Game class manages the teams added by holding a list of teams. The Team class manages players added to teams and maintains a list of players. The Player class creates the individual player object. This abstraction allows each system part to have what it needs without unnecessary details. Additionally, using lists and methods to add items to these lists will ensure unique game and team names.

## [Evaluation](#_2o15spng8stw)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | While Macs are not traditionally designed with server-side functions in mind, they can still be utilized for this purpose. Previously, Apple offered a server variant of its operating system known as macOS Server. However, this has since been discontinued, and support for earlier versions has been significantly reduced. Despite this, it’s still possible to host a web-based game on a Mac using server software such as Apache. This software comes pre-installed on macOS, but it requires proper setup and configuration to function as a server.  Advantages to using Mac to host the web-based game would be familiarity, built in software, and reliability. If you’re familiar with the Mac interface, you’ll find it user-friendly, which could simplify the setup and management of the server. Since macOS comes with Apache and PHP pre-installed, this could streamline the process of gathering the necessary software for the server. Macs are generally reliable and have smooth performance which could also be beneficial.  Weaknesses to choosing Mac to host a web-based game include scalability, limited software, and limited hardware. Without dedicated software for Mac to function as a server host, there will likely be problems when dealing with a large player count. Response times would be slow and the server itself could fail. Additionally, other services or software that might be needed at some point in the future might not be compatible or have very limited functionality on macOS. Finally, Macs generally don’t allow for upgrades or changes in hardware. Given the memory and CPU power needed for a server, this would significantly limit the capacity of the server. | The initial decision to be made is selecting the Linux distribution that will serve as the operating environment for the game server. Common selections include Ubuntu, CentOS, and Debian. Next, we would need to choose the web server software, with Apache and Nginx being popular options. The web server software will handle the requests from players accessing the game through their web browsers. Finally, we need to implement a database to store game-related data. Since the game is expected to work with thousands of players, we will need to use software such as MySQL, PostgreSQL, or MongoDB.  Advantages to using a Linux server are cost, security, and performance. Linux is open source which means that many of its resources, tools, and software programs are free. This will help reduce the overall cost and maintenance of the game application. There are paid versions with additional features for the distributions and database software that could incur costs. Linux has a good foundation of security and if a fix is needed there is usually one available. It’s also easy to customize security settings as needed to keep the system secure. Linux servers provide good performance and stability and can handle running more functions and features without issue since Linux is lightweight.  The biggest weakness with a Linux server is compatibility. If code or infrastructure already exists and needs to be migrated to Linux, that will take time. This is especially true if any part of the game application relies on Windows-based technologies. | When considering Windows for hosting a web-based game application, Windows Server OS is the first choice that comes to mind.  Windows Server is compatible with the .NET framework, which is likely to be used extensively if we are working with Windows, as many of the services and tools used throughout development will probably utilize this framework. This compatibility also facilitates easier integration with other Microsoft applications and services. Additionally, Windows Server allows for remote access, which can be advantageous for troubleshooting, updates and maintenance, monitoring, and providing flexibility in server management. Other advantages include security and support. The option of using Azure can add an extra layer of security, and Windows is generally quick to implement necessary security fixes. Extensive documentation is also available, which could be very useful. Furthermore, using Windows Server comes with the backing of Microsoft’s technical support if needed, and Microsoft generally provides quick driver support for new hardware.  Weaknesses to Windows Server OS include resource intensiveness, stability, and cost. Windows Server offers a wide array of services and features, but these require a substantial amount of RAM and higher-frequency cores on the CPU. Windows has also had some issues with memory leaks in the past, which could necessitate server reboots. In terms of cost, Windows Server is generally one of the more expensive options. The actual cost will depend on the number of players and games, which will determine the overall server scale. There are options to either purchase or lease a license. The current retail price for the 2022 edition of Windows Server ranges from $501 (for the Essentials version) to $6,155 (for the Datacenter version). | The design, hardware, and functionality of a mobile device does not lend itself to being a suitable server host for the game application. While it might technically be possible, there are no significant advantages to this approach, and it would likely not be worth the cost and effort required. This is especially true if the server needs to be capable of handling thousands of players. |
| **Client Side** | The client side of our gaming application would handle the rendering of the user interface, user input, communication with the server, performance, and client-side security. The browsers primarily used on Mac include Safari, Google Chrome, Mozilla Firefox, and Opera. On Mac systems, the primary browsers used are Safari, Google Chrome, Mozilla Firefox, and Opera. Safari is a browser that has been specifically built and optimized for Mac machines. However, its compatibility is largely limited to devices that run on macOS and iOS. Different browsers will have different rendering engines, and this needs to be accounted for to make sure that the website is displayed as intended.  Cost: In terms of cost, the use of HTML, CSS, and JavaScript, which are open source, would not directly incur any expenses. However, the primary costs would stem from the development and maintenance of the client-side of the application. Developers who will work on the UI design and developers who can handle both the front-end and back-end aspects of the website will need to be hired. As The Gaming Room is planning to outsource development, this could potentially lower the overall cost. However, the final cost will ultimately be determined by the allocated budget. In addition to these initial development costs, it’s important to factor in ongoing maintenance costs. These will be necessary to ensure the smooth operation and upkeep of the application over time.  Time: The timeline for the development will largely depend on the features of Draw It or Lose It. The features are not overly complex, but the development time will be determined by the expertise of the developers hired. With the right development team, this project could be completed within a timeframe of 1 to 3 months.  Expertise: Developers will be primarily working with HTML, CSS, and JavaScript for the front-end development. Java would be used for the back-end development. | The client side of our gaming application would handle the rendering of the user interface, user input, communication with the server, performance, and client-side security. The browsers primarily used on Linux include Mozilla Firefox, Opera, and Google Chrome. Different browsers will have different rendering engines, and this needs to be accounted for to make sure that the website is displayed as intended.  Cost: HTML, CSS, and JavaScript are open source and wouldn’t incur a cost, but the cost would come from development and maintenance of the client-side. We would need to consider developers who will work on UI design and rendering, and web developers who will work on the front-end and back-end of the site. Since The Gaming Room is looking to outsource development, the general cost will be lower but will ultimately be decided by the budget allocated. Additionally, maintenance costs need to be added to the total and these costs would be ongoing.  Time: The timeline for the development will largely depend on the features of Draw It or Lose It. The features are not overly complex, but the development time will be determined by the expertise of the developers hired. With the right development team, this project could be completed within a timeframe of 1 to 3 months.  Expertise: Developers will be primarily working with HTML, CSS, and JavaScript for the front-end development. Java would be used for the back-end development. | The client side of our gaming application would handle the rendering of the user interface, user input, communication with the server, performance, and client-side security. The browsers primarily used on Windows include Google Chrome, Mozilla Firefox, Microsoft Edge, and Opera. Of these browsers Microsoft Edge was built specifically for Windows machines although compatible with other operating systems. Different browsers will have different rendering engines, and this needs to be accounted for to make sure that the website is displayed as intended.  Cost: HTML, CSS, and JavaScript are open source and wouldn’t incur a cost, but the cost would come from development and maintenance of the client-side. We would need to consider developers who will work on UI design and rendering, and web developers who will work on the front-end and back-end of the site. Since The Gaming Room is looking to outsource development, the general cost will be lower but will ultimately be decided by the budget allocated. Additionally, maintenance costs need to be added to the total and these costs would be ongoing.  Time: The timeline for the development will largely depend on the features of Draw It or Lose It. The features are not overly complex, but the development time will be determined by the expertise of the developers hired. With the right development team, this project could be completed within a timeframe of 1 to 3 months.  Expertise: Developers will be primarily working with HTML, CSS, and JavaScript for the front-end development. Java, Ruby, and C# (if using Unity for image rendering) would be used for the back-end development. | The client side of our gaming application would handle the rendering of the user interface, user input, communication with the server, performance, and client-side security. On mobile devices, the primary browsers used are Google Chrome, Safari, Mozilla Firefox, Samsung Internet, and Opera Mini. Most of these browsers are modified versions of the desktop browser and can create differences in how the website is rendered. Furthermore, factors such as screen size, touch interface, and feature differences need to be considered when developing the game application. While the client-side development will generally be the same, adjustments may need to be made using CSS to modify layout, resize images, change font sizes, etc., based on the mobile device being used. Additionally, some features might not function as expected due to differences in input methods, hardware capabilities, and screen size.  Cost: HTML, CSS, and JavaScript are open source and wouldn’t incur a cost, but the cost would come from development and maintenance of the client-side. We would need to consider developers who will work on UI design and rendering, and web developers who will work on the front-end and back-end of the site. Since The Gaming Room is looking to outsource development, the general cost will be lower but will ultimately be decided by the budget allocated. Additionally, maintenance costs need to be added to the total and these costs would be ongoing.  Time: The timeline for the development will largely depend on the features of Draw It or Lose It. The features are not overly complex, but the development time will be determined by the expertise of the developers hired. With the right development team, this project could be completed within a timeframe of 1 to 3 months.  Expertise: Developers will be primarily working with HTML, CSS, and JavaScript for the front-end development. Java would be used for the back-end development. |
| **Development Tools** | On the server side, the development tools that would be utilized include Node.js, PHP, MongoDB or MySQL (for database management), and Socket.IO. Node.js would handle tasks such as managing the game logic, interacting with the database, handling user authentication, and facilitating real-time communication with clients. MongoDB or MySQL would be used to interact with the database that stores game-related data. Socket.IO is a JavaScript library for web applications that is compatible with all platforms, browsers, and devices and could be a valuable asset in the development process.  On the client side, the development tools being used are HTML, CSS, and JavaScript.  Programming languages used: HTML, CSS, JavaScript, and Java.  IDEs: While Xcode is Apple’s IDE there doesn’t seem to be plugin support for JavaScript. However Visual Studio does have a Mac specific edition that would best fit development needs. | For the server-side development, we could utilize tools like Node.js and PHP. Node.js enables JavaScript to run on the server, which could be used to manage game logic on the server, such as processing player guesses, updating the game state, and broadcasting updates to other relevant players. PHP, a server-side scripting language commonly used in website or application development, could be used to interact with the database or even dynamically generate HTML based on the game state. For instance, when a player makes a guess, a PHP script on the server could verify if the guess is correct, update the player/team score, and then generate an HTML response to send back to the player.  On the client side, we would use JavaScript, HTML, and CSS. JavaScript could be used to update the game state on the client side, handle user input, and communicate with the server. HTML and CSS would dictate the look and style of the website (including design graphics, colors, fonts, layout, etc).  Programming languages used: JavaScript, HTML, CSS, and Java. Also, the use of GitHub for version control should be used.  IDEs: Considering that the programming languages being used are geared towards web development and Java for game development, the best fit would likely be Visual Studio (or Visual Studio Code) or Eclipse. | Windows is compatible with a range of server-side technologies, including Node.js, Socket.IO, Express.js, and Redis. More specific to Windows are software such as ASP.NET, SignalR, SQL Server, Internet Information Services (IIS), and SmartfoxServer.  ASP.NET could be used to manage HTTP requests from players, route these requests to the appropriate game logic, and then send back HTTP responses. SignalR enables server code to send asynchronous notifications to the client-side. In our game this could be used to push updates from the server to the client in real time. An example for this would be if a player makes a guess, SignalR could be used to instantly update all other players with this new guess. SQL server could be used to store the game state, player information, and other persistent data. IIS serves as a backbone for web-based applications and accepts HTTP requests from clients (players) and routes the requests to the appropriate handlers (our game logic) and returns an HTTP response. Finally, SmartfoxServer provides a lot of features tailored to multiplayer games and would be very useful if more multiplayer game features were added in future editions of the game.  On the client side, technologies such as HTML, CSS, JavaScript, WebSockets, WebRTC, Unity3D, and React or Angular would be used. HTML could be used to structure the game interface, CSS could be used to style the interface, and JavaScript could be used to handle user input and communicate with the server. WebSockets could be used to send player inputs to the server and receive updates from the server in real-time. WebRTC uses simple APIs and could be used to possibly reduce latency. Unity3D could be used for the rendering of the images and would be able to accommodate any complexity of image added to the game. React or Angular are JavaScript libraries/frameworks and could be beneficial when building the user interface for the game.  Programming languages used: JavaScript, HTML, CSS, C# (if using Unity3D), and Java  IDEs: Visual Studio would be a good choice given its support and compatibility for most popular languages. Other options include Eclipse and NetBeans. | The client-side development of our gaming application would primarily utilize HTML, CSS, and JavaScript. CSS would be particularly important for adjusting how the website displays on a variety of mobile devices. For the back-end development, Java would be used.  Programming languages used: JavaScript, HTML, CSS, and Java  IDEs: While there are simplified IDEs that could run on a mobile device, it would be best to use a desktop computer for the development of the game and website. |

## Recommendations

1. **Operating Platform**:

The recommended operating platform to expand Draw It or Lose It to other platforms is Linux. Linux is secure and stable, capable of handling high traffic volumes and data without crashing or compromising user privacy. Furthermore, Linux’s compatibility with other operating systems allows users to access the game from any device equipped with a web browser, including Mac OS, Windows OS, and various mobile platforms. Linux also has free and customizable distributions and resources, which provide a reliable, flexible, and cost-effective solution for expanding Draw It or Lose It.

1. **Operating Systems Architectures**:

Linux, which is an open-source operating system, is compatible with a variety of processor architectures, including x86 and ARM. These architectures are prevalent in desktops/laptops and mobile devices respectively. Linux supports a range of web development tools and frameworks such as Apache, PHP, and Node.js, making it suitable for server use.

The Linux operating system contains key components such as the kernel and shell. The Linux kernel is the core component that manages hardware resources like CPU, memory, disk, and network. It also provides an interface for user applications to access hardware functionalities. The Linux shell is a command-line interpreter that enables users to interact with the operating system using commands (such as ls, mv, grep, etc.) and provides scripting capabilities for task automation and program execution.

For Draw It or Lose It, the Linux operating system architecture offers significant advantages and freedom in choosing both the software and hardware to be used. Linux’s scalability and flexibility allows it to handle high workloads and traffic by using multiple processors and cores. This will allow the game to be able to support many concurrent users and sessions.

1. **Storage Management**:

The Logical Volume Manager (LVM) could be an effective storage management system for Draw It or Lose It, which has a library of 200 high-definition images. LVM is a tool for Linux that enables the creation, resizing, and deletion of logical volumes. This storage virtualization allows for the optimization of storage space and game performance by dynamically allocating and reallocating disk space as required, without impacting the data on the volumes. Additionally, LVM supports snapshots, facilitating the creation of game data backups or enabling game restoration to a previous state in case of errors or failures.

For storage, a combination of hard disk drives (HDD) and solid-state drives (SSD) would be best. The HDD would be ideal for long-term data storage and less frequently accessed data. An SSD would be suitable for frequently accessed data and data that needs to be loaded quickly.

1. **Memory Management**:

Linux uses memory management techniques that could help Draw It or Lose It be efficient in its memory usage. Linux supports virtual memory, which allows the game to use more memory than the physical RAM available on the device. This enables the game to load and process large images without running out of memory. Linux also uses demand paging, which means that the game only loads the pages of memory that are needed at a given time. This reduces the memory footprint of the game and improves its performance. Additionally, Linux uses memory mapping, which allows the game to access files on the disk as if they were in memory. This could help the game’s performance and load time. Lastly, Linux uses copy-on-write, which means that the game can share memory pages with other processes until they are modified. This helps save memory and would allow the game to run multiple instances of itself without duplicating data.

1. **Distributed Systems and Networks**:

The Draw It or Lose It software must handle multiple components running on different platforms as all of them communicate over a network. The game consists of a web server that will host the game logic and the images, a database that would store the scores and player information, and a web browser (client) that will display the game interface and that the user will interact with. Since this is a web-based game, the network that will connect these components will be the internet.

To achieve communication between the components, the distributed software needs to use protocols for all messages exchanged. For example, the web server and the web browser may use HTTP as the application layer protocol, while the web server and the database could use SQL as the data access protocol. The network layer protocols, such as IP and TCP, would be responsible for routing and delivering the messages across the network.

The distributed software also needs to handle the dependencies between the components and the challenges of network communication. For example, the game depends on the availability and performance of the web server, the database, and the network. If any of these components fails or becomes slow, the game may not function properly or provide a satisfactory user experience. Therefore, the distributed software will need to implement mechanisms for fault tolerance, load balancing, caching, security, and synchronization.

1. **Security**:

Draw It or Lose It should use encryption to secure communication between the client and the server. Encryption will ensure that the data transmitted over the network cannot be intercepted or modified by unauthorized parties. HTTPS protocol, SSL/TLS certificates, and SSH keys should also be used. Additionally, the game should use authentication and authorization to verify the identity and access rights of the users. Authentication verifies that the user is who they claim to be, while authorization determines what the user can do on the platform. The game should use passwords or tokens for authentication, and roles, permissions, or policies for authorization.

Data protection also needs to be implemented to safeguard any user information stored on the server or the client. Data protection prevents the data from being accessed or altered by unauthorized parties. The game should use encryption, hashing, or salting for data protection, and backup, recovery, and deletion for data management.