

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 | 11/10/23 | Mark Baligad | Completed Executive Summary, Requirements, Design Constraints, and Domain Model sections. |
| 2.0 | 11/25/23 | Mark Baligad | Completed Evaluations and Recommendations sections. |
| 3.0 | 12/04/23 | Mark Baligad | Updated the Recommendations section. |

## [Executive Summary](#_sbfa50wo7nsh)

Creative Technology Solutions is supporting The Gaming Room in the development of a web-based version of the multiplayer Android game Draw It or Lose It. This design document evaluates the requirements, establishes design constraints, and then details the optimal development strategy to facilitate development in a web-based environment.

## Requirements

1. Creative Technology Solutions is creating the design document and initiating the development of the Draw It or Lose It app. The Gaming Room will use the framework provided to complete development.
2. A game will have the ability to have one or more teams involved.
3. Each team will have multiple players assigned to it.
4. Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
5. Only one game instance can exist in memory at any given time.
6. Hardware requirements will be established based on the software design.

## [Design Constraints](#_2et92p0)

1. The web-based game must support a wide range of devices and browsers to maximize the number of users that can access the game.
2. The Draw It or Lose It server-side application will be programmed in Java.
3. The Draw It or Lose It client-side application will be programmed in HTML and JavaScript due to JavaScript’s extensive support for browser-based applications, including multiplayer games.
4. Draw If or Lose It will be hosted on a server managed by The Gaming Room, which players will connect with to access the game.
5. Game progression and team management will be managed on the server side with user interaction occurring on the client side.
6. Network Bandwidth must support multiple users transmitting data without creating excessive delays that negatively impact player experience.

## [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 UML class diagram below outlines the relationship between classes in the Draw It or Lose It application. The Game, Team, and Player classes are inherited from the Entity class so that they can share basic identifiers such as a name and ID. The ID and name variables of each entity are encapsulated and made accessible by get methods. Encapsulation of unique IDs ensures that the application can effectively track individual instances of each Game, Team, and Player throughout the course of each game. The Team class contains a list of zero or more assigned players under a private list variable and can add additional players with the addPlayer function. The Game class contains a list of zero or more Teams and can add additional teams with the addTeam function. GameService is a singleton class that contains a list of zero or more Games and encapsulates tracking the unique IDs for each Game, Team, and Player. Use of the singleton design pattern ensures that conflicting instances of the GameService class are never created and that all games, teams, and players are tracked by a centralized service utilizing common memory. New Games can be created by the GameService addGame function. The ProgramDriver class executes the main function and uses the SingletonTester class to test that the GameService class only ever creates a single instance of the GameService.

**"The Gaming Room UML diagram. The top of the diagram is labeled as com dot gamingroom. Test boxes are placed in two layers. The first layer has three text boxes and the second layer has four of them. In the first layer, the 'ProgramDriver' textbox points to 'SingletonTester' textbox. The 'ProgramDriver' textbox contains the text 'asterisk main round brackets.' The 'SingletonTester' textbox contains the text 'asterisk testSingleton round brackets.' The arrow between these two text boxes are labeled 'open two angle brackets uses close two angle brackets'. In the second layer, there are 'GameService', 'Game', 'Team', and 'Player' text boxes. The 'GameService' textbox has texts arranged in two layers. The first layer contains games colon List open angle bracket Game close angle bracket, nextGamesId colon long, nextPlayer Id colon long, nextTeamId colon long, and service colon GameService. The second layer contains GameService round brackets, getinstance round brackets colon GameService, addGame open parenthesis name colon String close parenthesis colon Game, getGame open parenthesis id colon long close open parenthesis colon Game, getGame open open parenthesis name colon String close open parenthesis colon Game, getGameCount round brackets colon int, getNextPlayerID round brackets colon long, and getNextTeamId round brackets colon long. The 'GameService' box is connected with the 'Game' textbox with a line labeled 'zero dot dt dot asterisk'.  The 'Game' textbox also contains text in two layers. The first layers contains the text teams colon List open angle bracket Team close angle bracket. The second layer has Game open round bracket id colon long comma name colon String close parenthesis, addTeam open parenthesis name colon String close parenthesis Team, toString round brackets colon String. The 'Game' textbox is connected with the 'Team' textbox with a line labeled 'zero dot dt dot asterisk'. The 'Team' textbox also contains text in two layers. The first layers contains the text players colon List open angle bracket Player close angle bracket. The second layer has Team open parenthesis id colon long comma name colon String close parenthesis, addPlayer open parenthesis name colon String close parenthesis colon Player, and toString round brackets colon String. The 'Team' textbox is connected with the 'Player' textbox with a line labeled 'zero dot dt dot asterisk'. It contains the text Player open parenthesis id colon long comma name colon String close parenthesis and toString round brackets colon String. The 'Game', the 'Team, and the 'Player' boxes point to the 'Entity' textbox in first layer. The 'Entity' textbox contains text in two layers. The first layer has the text id colon long and name colon String. The second layer has Entity round brackets, Entity open parenthesis id colon long comma name colon String close parenthesis, getId round brackets colon long, getName round brackets colon String, toString round brackets colon String.**

## [Evaluation](#_2o15spng8stw)

Using your experience to evaluate the characteristics, advantages, and weaknesses of each operating platform (Linux, Mac, and Windows) as well as mobile devices, consider the requirements outlined below and articulate your findings for each. As you complete the table, keep in mind your client’s requirements, and look at the situation holistically, as it all has to work together.

In each cell, remove the bracketed prompt and write your own paragraph response covering the indicated information.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | * MacOS is a stable and reliable environment for a server. * MacOS hardware is expensive, * Hardware options are limited compared to Windows or Linux. * Licensing MacOS is expensive for commercial use. * MacOS license comes with technical support which would help in the event of issues. | * Linux is a cost-effective server solution due to being open source. * Linux is highly customizable and resource efficient working on a wide range of hardware. * Linux has a steeper learning curve in part due to its focus on command-line interfaces and open-source software. * Limited software availability compared to MacOS and Windows. | * Windows features user-friendly server software, cross-platform compatibility, and game hosting service support. * Licensing cost for Windows servers will increase with the number of players. * Windows servers support a range of different hardware. * Windows is considered more vulnerable to security issues. * Consumes more system resources compared to Linux. | * While it is technically possible to host a server on a mobile device, it is impractical due to hardware, operating system limitations, network availability, and the impact on device health from prolonged usage. * Applications or browsers on Mobile devices can communicate with servers to run/manage games. |
| **Client Side** | * MacOS supports a wide range of web browsers. * Modern web browsers on MacOS support JavaScript-based games. * No additional cost, time, or expertise is required to market to this audience. | * Linux supports a wide range of web browsers. * Modern web browsers on Linux support JavaScript-based games. * No additional cost, time, or expertise is required to market to this audience. | * Windows supports a wide range of web browsers. * Modern web browsers on Windows support JavaScript-based games. * No additional cost, time, or expertise is required to market to this audience. | * Modern web browsers on mobile devices support JavaScript-based games. * Additional cost and time will be required to ensure the game works on a mobile browser. * Games deployed on the Android or Apple App store will require review and approval. |
| **Development Tools** | * Java, HTML, and JavaScript based games for browsers on MacOS can be developed using a variety of different IDEs and development tools such as Visual Studio Code and WebStorm. * A wide range of Browsers such as Safari, Chrome, and Firefox can be used to test the game. * Eclipse and Visual Studio Code can be utilized on any PC environment for free Java development. * Some IDEs such as IntelliJ and WebStorm will require a paid license. | * Java, HTML, and JavaScript based games for browsers on Linux can be developed using a variety of different IDEs and development tools such as Visual Studio Code and WebStorm. * A wide range of Browsers such as Safari, Chrome, and Firefox can be used to test the game. * Eclipse and Visual Studio Code can be used for free Java development on any PC environment. * Some IDEs such as IntelliJ and WebStorm, will require a paid license. | * Java, HTML, and JavaScript-based games for browsers on Windows can be developed using various IDEs and development tools such as Visual Studio Code and WebStorm. * A wide range of Browsers such as Safari, Chrome, and Firefox can be used to test the game. * Eclipse and Visual Studio Code can be used for free Java development on any PC environment. * Some IDEs such as IntelliJ and WebStorm, will require a paid license. | * Java, HTML, and JavaScript-based games for browsers on mobile devices can be developed using various IDEs and development tools such as Visual Studio Code and WebStorm. * A wide range of Browsers, such as Safari, Chrome, and Firefox, can be used to test the game. |

## 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 is the recommended Operating System to host the game server. Compared to other options, such as Windows Server, the lack of licensing fees for Linux will contribute to significant savings. The proven stability, security, performance, and cost savings make Linux the best browser-based game server. Linux is also compatible with Amazon Web Services and enables a serverless architecture for a low-cost and scalable server solution.
2. **Operating Systems Architectures**: Linux is built on a kernel model, where the kernel manages system resources, hardware, network connections, and software processes. Linux and AWS utilize role-based user permission systems for access control in addition to multiple layers of security and monitoring. Open-source Linux distributions that will reduce costs and offer high flexibility and customization compared to pricey Windows and MacOS Servers are widely available. Popular Linux distributions such as Ubuntu Server have a broad user base, meaning their extensive documentation and support is available to ensure smooth development and maximum uptime. Linux is also known for its security features, resource efficiency, hardware compatibility, and scalability, which will aid The Gaming Room in supporting the most extensive player base possible compared to a MacOS or Windows server.
3. **Storage Management**: The recommendation for Storage Management is to utilize cloud storage. Amazon Web Services offers the least expensive option and maintains an advantage over other alternatives such as Microsoft Azure and Google. A cloud storage provider service such as Amazon Simple Storage Service offers free tiers that will reduce costs during testing and allow for scaling as the application and player base storage requirements increase. Running the application from a container through a cloud storage system such as Amazon ECS eliminates the cost of maintaining physical server hardware and employing full-time support personnel.
4. **Memory Management**: The Draw If or Lose It software will use random-access memory (RAM) to store information and feed it to the central processing unit (CPU) while the game service is active. Data will also be held on the hard drive for long-term storage, so data about Games, Teams, and Players can be retained if the application is closed or the server is reset. Linux offers lightweight distributions such as Ubuntu Server, which offers efficient memory and resource usage and has a broad user base with extensive documentation. Linux supports a 64-bit architecture which can handle large amounts of RAM to ensure fast operation and stability. The Linux kernel offers greater flexibility regarding parameters that can be used to optimize memory management than systems such as Windows or MacOS. An Amazon container automatically reserves memory and system resources needed to support the application. Automatic scaling of memory reservation to accommodate increases in the player base offers significant advantages in terms of cost and uptime involved in purchasing, installing, and maintaining requirements for physical hardware.
5. **Distributed Systems and Networks**: A Linux server running over Amazon Web Services (AWS) will host the Java-based Game Service and act as the hub for clients connecting through their browsers. The clients will execute HTML and JavaScript code in their local environment, which displays data to the user and communicates with the server using REST API calls. In the Linux server, data is received from the clients, processed, stored, and then distributed back to the clients. Amazon instances allow for large amounts of inbound and outbound traffic, which can scale depending on the application’s plan and number of vCPUs. AWS ensures consistent uptime to ensure the application is accessible to users with service credits should service ever drop below their 99.9% service level agreement. Outsourcing the network system allows the Gaming Room to focus on the application and limits costs associated with managing network maintenance and diagnostics in-house.
6. **Security**: In the application, users will create a username and password to restrict access to their accounts. The password will be encrypted and stored on the server so that the passwords are not exposed if the data is exposed. The client and server will use a secure HTTPS connection to encrypt data transferred between the browser and server to protect user information. Access to specific API calls, such as viewing a list of all users, will be restricted to administrator roles, which regular users will be prohibited from. Access control on the server will limit access to only administrators to protect user data. All user data will be encrypted on the server to prevent attackers from being able to view or steal sensitive information. The application will utilize input validation to prevent attackers from injecting malicious code or executing buffer overflows. The use of a firewall and secure protocols on the Linux Server will prevent unauthorized access and allow administrators to control network traffic through specified ports. Amazon Web Services operates on a Virtual Private Cloud that isolates resources and safely routes network traffic through secure subnets. Amazon’s AWS Network Firewall protects against unauthorized access, such as malware and DDoS attacks.