

# Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.4

## 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 4**](#_Toc115077324)

[**Evaluation 5**](#_Toc115077325)

**Recommendations 10**

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 05/20/24 | Brad Wells | Initial Document Creation |
| 1.1 | 05/25/24 | Brad Wells | Updates to Design Constraints and Domain Model |
| 1.2 | 06/09/24 | Brad Wells | Updates to Evaluation |
| 1.3 | 06/12/24 | Brad Wells | Revisions to Evaluation |
| 1.4 | 06/16/24 | Brad Wells | Updates to Recommendations |

**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)

Creative Technology Solutions is seeking to develop a web-based version of The Gaming Room's popular Android app, Draw It or Lose It. The aim is to create a versatile gaming application that can operate across multiple platforms, ensuring a consistent user experience. This document outlines the software design for the new game, addressing critical requirements such as team and player management, unique naming conventions, and maintaining a single game instance in memory. The proposed solution involves utilizing web technologies and a distributed architecture to meet these requirements effectively.

## Requirements

Business:

Technological:

## [Design Constraints](#_2et92p0)

Platform Independence: The game must be accessible on multiple platforms, including web browsers, Android, and iOS devices. This constraint necessitates the use of responsive design principles and cross-platform technologies such as HTML5, CSS3, and JavaScript frameworks like React or Angular. Utilizing these technologies ensures that the game renders correctly on different devices and screen sizes, providing a consistent user experience.

Concurrency Management: Only one instance of the game should exist in memory at any given time, even with multiple teams and players interacting simultaneously. This constraint implies the need for efficient session management and unique identifier generation for games, teams, and players. Implementing a back-end system with a scalable architecture that can handle multiple users while maintaining the integrity of the single game instance requirement is a necessity.

Unique Naming Conventions: Game and team names must be unique to prevent conflicts and ensure smooth gameplay. This constraint requires implementing a name verification system to check for name availability in real-time.

Real-time Interactivity: The game involves real-time drawing and guessing, requiring a responsive and low-latency environment. This constraint requires the use of technology that will enable real-time communication between the server and clients.

Security and Scalability: As a web-based application, the game must ensure data security and scalability to handle a growing number of users. Implementing secure authentication mechanisms and data encryption will protect user data and game integrity. Regular audits of these systems will be critical to the maintainability of 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 UML diagram below represents The Gaming Room's application for Draw It or Lose It, incorporating object-oriented programming principles such as inheritance, encapsulation, composition, and design patterns like Singleton. These principles are utilized to meet the customer requirements, ensuring a scalable and maintainable application architecture. The diagram provides a blueprint for developing a web-based version of the game, facilitating multi-platform compatibility, and streamlined game management. Inheritance is seen in the diagram by the Entity class, which is a base class for Player, Team, and Game. This allows these classes to share common attributes and methods, promoting code reuse and reducing redundancy. Encapsulation is demonstrated by each class encapsulating its data (attributes) and providing public methods to interact with this data, ensuring that the internal state is protected from unauthorized access or modification. Composition and aggregation are exhibited by The Game class, containing a list of Team objects, and the Team class contains a list of Player objects. This demonstrates the has-a relationship, which is a form of composition/aggregation that models the whole-part relationship. A singleton pattern is also used by the GameService to ensure there is only one instance of the service managing all game-related operations, ensuring state management across the application. A breakdown of the classes in the game code and UML Diagram follow:

Entity Class:

This is a base class that holds common attributes (id, name) and behaviors that are inherited by other classes (Game, Team, Player). This class demonstrates inheritance and encapsulation, allowing shared properties to be centralized and reused.

Player Class:

Represents individual players within a team. Inherits from Entity, thus gaining its common attributes and methods.

Team Class:

Represents a team that consists of multiple players. This class uses composition to hold a list of Player objects, demonstrating aggregation. It also inherits common attributes and methods from Entity.

Game Class

Represents a game instance with multiple teams. This class uses composition to manage a list of Team objects and inherits common properties from Entity.

GameService Class:

Provides services related to game management, such as adding and retrieving games. It employs the Singleton pattern to ensure only one instance of GameService exists. This class also demonstrates the Factory pattern by generating unique IDs for games, teams, and players.

ProgramDriver Class:

Serves as the entry point for the application, demonstrating how the application uses the GameService to manage the game lifecycle.

SingletonTester Class:

Used for testing the Singleton pattern implementation in 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** | Mac servers are built on Unix, which is a stable and secure environment. macOS is known for its user-friendly interface and integration with other Apple products.  Advantages: Integration with iOS development environments, strong security features like sandboxing and Gatekeeper, and Unix-based architecture for stability.  Weaknesses:  Mac is less commonly used as a server platform which limits support for it. Higher cost of hardware and limited scalability compared to other server environments, smaller community support, and fewer enterprise solutions. | Linux is an open-source operating system known for stability, security, and flexibility. It supports a wide range of server applications.  Advantages: High customizability, extensive community support, cost-effective, security features, and compatibility with a wide range of hardware.  Weaknesses: Requires more technical expertise to manage and configure, can be complex for users unfamiliar with command-line interfaces. | Windows Server is a widely-used commercial server operating system known for its ease of use and integration with other Microsoft products.  Advantages: User-friendly GUI, strong enterprise support, integration with Active Directory and other Microsoft services, extensive software compatibility.  Weaknesses: Licensing costs can be high, potentially less secure compared to Unix-based systems, higher resource consumption, and offer less customization than Linux. | Mobile devices typically run on iOS or Android, with limited server capabilities due to hardware constraints.  Advantages: Portable, can leverage mobile network connectivity, useful for edge computing scenarios.  Weaknesses: Limited processing power and storage, battery life constraints, not suitable for hosting high-traffic web applications. |
| **Client Side** | Development for Mac clients often requires knowledge of macOS-specific APIs and tools like Xcode as well as expertise in Swift or Objective-C languages. The cost of development can be higher due to the need for Apple hardware.  When considering time, the cost is moderate, depending on familiarity with Apple's ecosystem.  Required expertise is relatively high, specific to macOS development. | Developing for Linux clients involves using tools like GCC, Make, and IDEs like Eclipse or Visual Studio Code. Familiarity with languages like C, C++, Python, or Java is beneficial.  Costs are relatively low, with open-source tools and environments.  Time is variable, depending on the developer's familiarity with Linux.  Expertise required tends to be moderate to high, depending on specific distributions and environments, which can be a limiting factor when implementing a Linux server.  Linux is also known for being very customizable, allowing for an organization to manage security very easily. | Developing for Windows clients typically requires tools like Visual Studio and knowledge of languages like C#, .NET, or C++. The development environment is user-friendly, but licensing costs can be prohibitive.  The time required is moderate with extensive documentation and community support available with a moderate level of expertise required in Windows development. | Mobile development for iOS and Android requires specific knowledge of Swift/Objective-C for iOS and Kotlin/Java for Android. Tools like Xcode and Android Studio are essential.  Associated costs are high for iOS (Apple hardware) and moderate for Android.  Time required is relatively high, considering different platforms and testing.  A high level of expertise is required with specific knowledge for each mobile OS. |
| **Development Tools** | Developing game applications for Mac utilizes programming languages such as Swift, Objective-C, and C++, along with game engines like Unity for C# and Unreal Engine (C++). Essential tools include Xcode, Unity, and Git (for version control). The development team must have expertise in Swift and Objective-C for macOS-specific development, potentially needing a dedicated team proficient in these languages. While Unity and Unreal Engine can standardize development across multiple platforms, optimization for macOS might still require specialized knowledge. Mac also supports the use of frameworks like React and Flutter for cross platform development. Licensing costs include a free Xcode IDE, but a $99 per year Apple Developer Program membership is required for App Store publishing. Unity has a free tier with professional versions incurring fees. | On Linux, this involves C++, Python, and Java, along with game engines, Unity and Unreal Engine. The primary tools include Eclipse, Visual Studio Code, CLion, Unity, Unreal Engine, Blender, and Git. When it comes to managing dependencies, package managers that can be used are APT and YUM. The development team needs expertise in C++ and familiarity with open-source development practices. Unity and Unreal Engine support can facilitate cross-platform development but require knowledge of C# or C++. Additional expertise may be needed for system configuration and optimizing game performance on different Linux distributions. Most development tools and IDE’s for Linux are free and open-source, with Unity and Unreal Engine licensing ranging from free to paid professional versions. | Windows is an excellent choice for server-side operations in the enterprise environment. Developing for Windows involves using programming languages such as C#, C++, and Java. Tools include Visual Studio, Unity, Unreal Engine, Blender, and Git. The development team benefits from access to a wide range of tools and extensive documentation available for Windows, making it easier for a single team to handle various aspects of development. Cloud solutions are offered with Microsoft Azure and database with SQL Server. Licensing costs can be higher than using Linux and lack customization. Expertise in C# and C++ is crucial, particularly for using DirectX and popular game engines. Large-scale projects may still require specialized teams for graphics, networking, and performance optimization. Licensing costs include a free Visual Studio edition, with enterprise versions requiring a subscription. Unity and Unreal Engine licensing is the same as Mac/Linux. | Mobile devices on iOS use Swift or Objective-C while Android utilizes Kotlin, Java, Unity, and Unreal Engine. Xcode, Android Studio, Unity, Unreal Engine, Blender, and Git are the primary tools used in development. The development team needs expertise in both iOS (Swift/Objective-C) and Android (Kotlin/Java) development. Cross-platform development tools like Unity and Unreal Engine can streamline the process but platform-specific optimizations and testing are still required. Separate teams may be required to handle different aspects of mobile game development, such as UI/UX design, backend services, and performance optimization. Licensing costs for mobile app development include a free Xcode IDE but a $99 per year membership is needed for App Store publishing. On Android, the Android Studio IDE with a one-time $25 Google Play Store registration fee, makes it more accessible than iOS. Unity and Unreal Engine licensing is the same as across all platforms. |

## 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**:

Based on evaluations, I recommend the Linux operating system for hosting Draw It or Lose It. Linux is a cost-reducing, open-source OS, supports various development tools, and is both flexible and secure. The stability and scalability offered by Linux make it an ideal choice for a high-traffic, multi-user environment, as required by The Gaming Room.

1. **Operating Systems Architectures**:

All operating systems have similar architectures, consisting of hardware, kernel, and shell components. The hardware layer includes essential physical components such as memory, CPU, input/output devices, and other peripheral devices. This layer is the foundation upon which the entire operating system operates, providing the necessary resources for the system's functionality. Linux is an ideal OS because it can run on a variety of hardware offering more flexibility and potential for reduced costs. At the core of the Linux operating system is the kernel, the component that acts as a bridge between applications and the hardware. The kernel manages system resources, including memory, processes, and devices, allowing all to operate without cohesively. It handles critical functions such as process scheduling, memory management, device management, and system calls, acting as the primary enabler for communication between software and hardware. The shell in the Linux architecture provides an interface for users to interact with the operating system. It interprets user commands and scripts, facilitating communication with the kernel to execute tasks. The shell can be command-line based or graphical, offering flexibility in how users interact with the system, depending on user proficiency. The modular design of the Linux architecture, with its distinct separation between hardware, kernel, and shell, contributes to its flexibility, stability, and security as the operating system of choice for Draw It or Lose It.

**Storage Management**: It is recommended that cloud storage be utilized. While a subscription to a cloud service provider is a ongoing and additional cost, it will be beneficial in the long run. The cloud allows the application to only use the storage space needed, with the ability to scale as the game's popularity grows. Google Cloud Platform (GCP) the best fit for this application due to its scalability, redundancy, and security features. Cloud storage provides flexibility and cost-effectiveness, eliminating the need for significant upfront hardware investments that will inevitably need to be replaced or upgraded, hindering scalability in the future.

1. **Memory Management**: Linux has efficient memory management capabilities. For this game, Java is used as the primary backend development language. Java handles memory management automatically through its garbage collection system, reducing the need for manual memory management and minimizing the risk of memory leaks. This task is performed by sorting and iterating through data heaps which is where objects are allocated during runtime. Its main job is to identify and reclaim memory occupied by objects that are no longer needed or referenced by the program to free up memory for new allocations. This dynamic memory management ensures optimal performance and stability of the game.
2. **Distributed Systems and Networks**: To ensure Draw It or Lose It communicates effectively across various platforms, I recommend a distributed architecture with a focus on web-based access. Google Chrome, available on all major platforms, will serve as the primary browser for accessing the game. This simplifies compatibility and ensures that users can play the game regardless of their operating system. Key components of this architecture include: microservices, API gateways, and networking protocols. Microservices enable independent scaling and resilience, reducing the likelihood of outages. API gateways manage and secure traffic acting as a single-entry point for all client requests to microservices. Networking protocols ensure efficient and real-time communication through HTTP/HTTPS. In the case of Draw It or Lose It, the Web Socket API is also recommended, providing a persistent connection between the client and server, allowing instant data exchange without the need for repeated HTTP requests.
3. **Security**: Security is always important in the ever-increasing cyber threat environment. Utilizing Google Cloud services provides security for both data at rest and data in transit. Key security measures include encryption, access controls, audits, and quality assurance testing. Data encryption offered by GCP is AES-256 for data at rest and TLS (Transport Layer Security) for data in transit. GCP has an Identity Access Management service that allows for managed access to resources by assigning roles and permissions (RBAC) to users and service accounts. This ensures that only authorized users can access and modify your data and services. Multi-factor authentication can also be implemented as an additional layer of security. Regular logging is conducted by GCP, allowing for routine audits and monitoring of the application data with alerts for potential security incidents. Quality assurance testing is critical prior to deployment of the application to ensure there are no vulnerabilities in security. GCP is also a reputable service that complies with industry standards and regulations for data protection.