

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 | 03/23/2024 | Karina Aronov | Executive Summary, Design Constraints, Domain Model |
| 1.1 | 04/04/2024 | Karina Aronov | Evaluation |
| 1.2 | 04/15/2024 | Karina Aronov | 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)

The Gaming Room wants to create an online game called “Draw It or Lose It” that will run on different operating systems. Right now, it is only available on Android. Several teams will be competing to guess which drawings are being created in this game. It must have the capacity to support many teams, each with multiple players. Since there can only be one instance of a game running in memory at once, game names and team names need to be unique. The game will also have to maintain the fairness of the game for all players. In order to ensure that every aspect of the game, from guessing to drawing, occurs in real-time, we plan on using advanced software to oversee the game's internal operations. To keep the game entertaining and fast-paced, we'll also make sure it can support a large number of players without becoming slowed down.

## Requirements

* A game will have the ability to have one or more teams involved.
* Each team will have multiple players assigned to it.
* Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
* Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.

## [Design Constraints](#_2et92p0)

* The game will allow for multiple teams to compete.
* Multiple players per team.
* Verification of player names and team names to ensure the names are a unique identifier.
* Only one instance of the game can be opened at a time.
* Each game will consist of 4 rounds, each round will be 1 minute long.
* Drawings will be generated over the course of 30 seconds until it is fully revealed.
* Each team has until the timer ends to guess the drawing, otherwise all other teams will have 15 seconds to guess.
* Web-based version.

## [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 shows us the structure of how we will develop the gaming software. The ProgramDriver class includes the main method which is the entry point and will run the application. It uses the SIngletonTester to ensure that only one instance of GameService is active. The GameService class is the most important class because contains the majority of the methords necessary to run the program and its functionalities. GameService also manages the Game, Team, and Player classes. They all represent different components of the game while sharing common features they inherit from the Entity class. The Entity class is the parent class to these classes. The open arrow shows us that this is an example of inheritance. The Game, Team, and Player classes along with the GameService class show us that they all share the same multiplicity which means they may contain zero to many objects. Associations between the classes reflect the relationships of the game’s components, allowing the software to mimic the gameplay structure.

**"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** | **Characteristics:**  MacOS offers strong performance and a more stable environment to prevent viruses.  **Advantages:**  The best platform for creating apps for iOS and macOS. Seamless ecosystem.  **Weakness:**  Mac is expensive and has limited sever hardware. | **Characteristics:**  Linux is the industry leader in web servers. Supports a large variety of server applications.  **Advantages:**  Compatible with a wide range of sever software. Highly scalable and cost effective.  **Weakness:**  Can be difficult to learn if you are unfamiliar with Unix-based systems. Also requires more hands-on management. | **Characteristics:**  Windows is user friendly and offers full integration with other Microsoft services.  **Advantages:**  Ideal for those who are invested in Microsoft. Strong support for .NET applications.  **Weakness:**  More expensive due to licensing fees. Fewer open-source server software options. | **Characteristics:**  Can be used to access web apps but it is not practical for production due to hardware and connectivity limitations.  **Advantages:**  Cloud services allow for high scalability. Cost efficient. Offers a wider reach.  **Weakness:**  Not suitable for server-side hosting. Restricted memory, CPU, and network bandwidth. |
| **Client Side** | **Software Development Considerations:**  Can be costly as developing for Mac clients can require investing in both software and hardware. Xcode offers quick development but requires knowledge. | **Software Development Considerations:**  Linux offers open source which allows for a wide variety of development tools and libraries at no cost. No need for specific hardware except for a PC that can run Linux. Requires knowledge of Linux and a wide range of programming languages. | **Software Development Considerations:**  Development can be efficient with the use of Visual Studio, Atom, or Eclipse. Windows follows standard web practices. Cost depends on the type of setup but can be affordable. | **Software Development Considerations:**  Development requires considering different screen sizes. App stores require a review process which can stall launch. Memberships are costly for developer programs. |
| **Development Tools** | **Tools:**  Xcode for macOS and iOS applications. Visual Studio, Atom and JetBrains IDE for web and cross-platform.  **Languages:**  HTML, Java, CSS, Swift. | **Tools:**  Eclipse, Visual Studio, and JetBrains IDEs. Native Linux tools like GCC for C/C++ development.  **Languages:**  Python, Java, C/C++ | **Tools:**  Visual Studio for .NET, C#, and web & cross platform. JetBrains IDEs.  **Languages:**  C#, .NET, JavaScript, C++, Python. | **Tools:**  Xcode for iOS and Android Studio for Android. React Native and Flutter for cross-platform frameworks.  **Languages:**  Swift (iOS), Kotlin (Android), Dart (Flutter), JavaScript (React Native). |

## 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**: I think Linux would be the best recommendation for hosting the server-side components of "Draw It or Lose It”. Linux offers scalability and flexibility, which are important for managing the large number of ongoing game sessions and real-time data transfers. Because of its affordability, more funds can be spent on game development and improvement as opposed to infrastructure. There is also a wide support for various programming languages and server technologies, which is beneficial for a game expected to run in diverse environments and potentially scale significantly. Advanced multiplayer games requiring high availability and dependable real-time interactions are ideally suited for Linux because of its adaptable structure and strong performance in web server setups.
2. **Operating Systems Architectures**: Linux supports a wide range of hardware from mobile devices to mainframes, making it an extremely flexible option for several application demands. Linux architecture is known for its excellent performance, security, and adaptability in networked contexts, all of which are essential for server-side multiplayer game operations. Linux distributions can be modified to contain only the parts needed for particular tasks, which lowers system overhead and improves performance.
3. **Storage Management**: “Draw It or Lose It” would not require too much storage, as it would use roughly 1.5GB to 2GB for images and additional space for the app’s executable files, data storage, and potential future updates. Using WebP file type for the images would offer high compression rates without much loss of quality. Additionally, using a storage system such as AWS would offer major benefits, especially in terms of scalability, reliability, and security. AWS services' auto-scaling features let you adjust storage capacity in relation to demand, which is essential for games that experience varying user base. AWS provides an extremely durable storage frame intended for primary data and highly important data storage. Data is stored in multiple places on several devices across various facilities. Data is also secured with encryption whether it is in transit or at rest. Lastly, AWS has data centers globally, which helps deliver content quickly and with low latency to a global player base, improving the user experience (*Cloud Computing Services - Amazon Web Services (AWS)*, n.d.).
4. **Memory Management**: Linux's advanced memory management system is great for the demands of the game as it requires reliable real-time processing and the ability to handle many tasks at once. It uses techniques such as dynamic memory allocation to maximize RAM utilization and reduce waste, as well as process isolation to prevent various components of the system from interacting with one another. Linux controls virtual memory by means of paging and controlled swapping. It applies advanced algorithms to guarantee that the most frequently used data is always easily accessible, preventing any interruptions caused by excessive memory swapping. Linux boosts performance with its caching mechanisms, which speeds up the loading times by reducing the need to read data from the disk repeatedly (*Chapter 3 Memory Management*. (n.d)).
5. **Distributed Systems and Networks**: Networks and distributed systems can become complicated due to managing issues like delays, keeping data in sync, and handling network disruptions, all of which can be unpredictable and affect the system's overall function. (*Challenges With Distributed Systems*, n.d.) so I would suggest AWS for the distributed system and network as well as their storage management. AWS is the strongest web service provider, and they offer dependable connectivity allowing for efficient management of distributed resources. This ensures the system is both strong and quick enough to handle varying game traffic and network conditions seamlessly. This leads to a more reliable and enjoyable gaming experience for all users. They use a system where server tasks are split into smaller, specialized sections, known as microservices. This setup allows each part of the game, such as managing player sessions or processing gameplay to operate independently (*Distributed Systems Components - Implementing Microservices on AWS*, n.d.). This flexibility means the game can handle changes in player numbers more easily, scaling up or down as needed without affecting overall performance. It also makes updating the game simpler, as changes can be made to one part without disrupting others. AWS also has the ability to automatically scale which can come with both its pros and cons. Automatic scaling is beneficial because it monitors your applications and automatically adjusts capacity to maintain steady, predictable performance but it can become costly if more and more resources are needed (*Application Scaling - AWS Auto Scaling - AWS*, n.d.).
6. **Security**: In order to guarantee strong security for "Draw It or Lose It," it is essential to put extensive data security protocols in place. Sensitive information is protected from unwanted access and interception by encrypting all user data, both in transit (using SSL/TLS) and at rest (using strong encryption techniques). Role-based access controls and multi-factor authentication strengthen the security of user accounts by limiting access according to designated roles and responsibilities. In order to protect the network from potential dangers and unauthorized access, it is helpful to set up firewalls and intrusion detection systems. Preventing vulnerabilities in the system requires careful management and frequent upgrades.

References:

*Application scaling - AWS Auto scaling - AWS*. (n.d.). Amazon Web Services, Inc. https://aws.amazon.com/autoscaling/

*Challenges with distributed systems*. (n.d.). Amazon Web Services, Inc. https://aws.amazon.com/builders-library/challenges-with-distributed-systems/

*Chapter 3 Memory Management.* (n.d)

https://tldp.org/LDP/tlk/mm/memory.html

*Cloud computing services - Amazon Web Services (AWS)*. (n.d.). Amazon Web Services, Inc. https://aws.amazon.com/

*Distributed systems components - Implementing Microservices on AWS*. (n.d.). https://docs.aws.amazon.com/whitepapers/latest/microservices-on-aws/distributed-systems-components.html