

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 7**

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 5/30/2024 | David Wasson | This is the first iteration of a proposed solution for creating a web-based game called Draw It or Lose It. |
| 2.0 | 6/16/2024 | David Wasson | Further evaluation of platforms to use for the web-based game called Draw It or Lose It. |
| 3.0 | 6/30/2024 | David Wasson | Added recommendations for the Gaming Room. |

**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)

This software design document was designed to help outline a proposed solution for the development of a web-based game called Draw It or Lose It for the client The Gaming Room. The game itself is loosely based on a classic tv game show Win, Lose, or Draw. Competing teams try to guess what is being drawn on their screen. The Gaming Room has an app version of the game already available, however, they are wanting to develop a web-based platform for the game to reach more players and be available on more devices. The client isn’t sure where to start in creating this new environment and has enlisted my help with preparing this software design document to propose a way to accomplish their goals.

## Requirements

1. The game must have the ability to have one or more teams involved.
2. Each team will have multiple players assigned to it.
3. Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
4. Only one instance of the game can exist in memory at any given time.
5. During the game, each round should last for one minute, with the drawings being completed at the 30-second mark.
6. If a given team fails to guess correctly what the drawing is within their given time, the other teams will get a chance to provide their guess in no more than a 15 second time limit.

## [Design Constraints](#_2et92p0)

1. Only one instance of the game can be in memory at any given time.
2. The system must check for unique game and team names.
3. The game will be web-based. This means it should be compatible with multiple browsers and multiple devices.

## [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 outlines all the classes and how they work together in the game Draw It or Lose It. We see that the Game, Team, and Player classes extend the Entity class. Each of these subclasses contains an id and name that helps with making sure only one game is in active memory at a time. These classes also contain all the information related to each game instance, including player names, team names, and the game name. The GameService class helps manage game instances as well by obtaining the information from the subclasses Game, Team, and Player. Then it checks those against the current game instance that is stored to determine if this instance is new or if it already exists in the memory of the game. Within the ProgramDriver class, the main function is listed. This is the starting point for the game application. Once the application is started and the ProgramDriver class is into effect, then the SingletonTester class is created. This is what tells the game that only one instance can be active at a time. Once the information for the game is obtained from the GameService class, it is checked through the SingletonTester class to ensure this is a new or existing game, allowing the functionality desired by the client. Depending on what is found either the old instance is loaded, or a new instance is created while removing the old instance. Then the ProgramDriver class is allowed to proceed. The UML diagram does indeed show some object-oriented programming principles. First, we see inheritance. The Game, Team, and Player classes inherit the id and name attributes from the Entity superclass. The GameService class shows encapsulation by gathering attributes or the information provided in the other classes. Then proceeding to have that data checked for existing games without showing all the background processes to the player. This ensures privacy of the data while also making sure only one instance of the game is created. This also identifies abstraction in the setup. By not showing the players the background processes and focusing on what is needed, the teams and players only see what they need to see when playing the game.

**"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**  Limited availability for users of the game. Additionally, we would need to consider that we would need to pay for the Leopard Server which offers an unlimited-client license. At $999, the cost is steep. | **Linux**  Steady platform, but some users may not be familiar with it. Costs associated for a server with this platform can go up to $1300. The cost can be reduced if we choose to self-support and maintain the server. | **Windows**  Familiarity and being user friendly are key. The compatibility offered is another positive. However, this could be a costly route. To fit the needs, there could be costs up to $6,155 for a license. | **Mobile Devices**  This option is less optimal for also having a web-based game. The platform and server needs are too large. The performance would suffer. Costs for a server here could be thousands of dollars depending on who hosts. |
| --- | --- | --- | --- | --- |
| **Server Side** | Very secure system that is reliable. Great support team and graphics.  However, limited in customization and can become costly to maintain. Also is limited to certain devices. | Great for customization as it is compatible with many different tools. Has good reliability and security features. Can be used by most devices. Limited with graphics. | Compatible with many different tools and software systems that allow for great customization. Great support system and compatible with most devices. However, less secure and stability could become an issue. | Great for portability and support resources. However, graphics may suffer, screen size could cause issues, and may not be compatible with many devices. |
| **Client Side** | User friendly with a clean interface. Some users will be able to identify with how to use this. Responsive to user commands. Limited in the devices it can be used on. May require updates that increase costs. | While free to use, it may not be as user friendly. Users may have a hard time learning to use the system. | Most users will be comfortable with this system. However, could be costly with multiple updates. | Design limitations and responsiveness are less than desirable. May not be user friendly and could be limited in devices that are compatible. |
| **Development Tools** | Compatible with multiple languages, including Java, HTML, and CSS. Can use PyCharm and Visual Studio. | Compatible with Java, CSS, and HTML. Can use PyCharm and Visual Studio among others. | Compatible with Java, Python, C#, and .NET among others. Can use PyCharm, Visual Studio, and Eclipse among others. | Compatible with Java, JavaScript, Kotlin, and Swift. Can use Android Studio, Xcode, and Visual Studio. |

## 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: For the client The Gaming Room, my recommended operating platform would be to use a cloud-based operating platform for the Draw It or Lose It application. There are multiple reasons as to why I would recommend a cloud-based operating platform. Cloud-based platforms have superior scalability compared to traditional systems. This will allow the application to grow, be modified, and keep up with multiple users. Being cost effective is another reason to like cloud-based platforms. When comparing the costs of upgrading and maintaining a server, cloud-based platforms come out ahead of traditional systems. Performance of a cloud-based platform is also better compared to traditional platforms. A cloud-based platform doesn’t have the same limitations that traditional platforms do.
2. Operating System Architectures: Cloud-based platforms have architectures that make it the better choice for Draw It or Lose It. Abstraction is a key component of a cloud-based platform This helps maximize resource utilization while multiple users connect and interact with the application. Cloud-based platforms provide scalability, which is essential for performance, maintenance, and growth of the application. Service disruptions tend to be cut down with cloud-based platforms as they have a dedicated team to monitor and maintain the server for the client. Implementing updates and new features goes smoother and quicker with a cloud-based platform.
3. Storage Management: When it comes to storage management, Amazon S3 can’t be overlooked. This would be my recommendation for Draw It or Lose It. I chose this because of the scalability and security this option provides. There is practically no limit to the amount of data you can store using this storage service. This allows the game to grow, evolve, and provide users with a great experience. Security is essential to any application that wants to succeed. Users will not want to continue to use the application if they are at risk any time they do so and Amazon S3 provides active monitoring, encryption, and alerts when issues are found. This again allows the user to have a great experience.
4. Memory Management: Memory management is important when it comes to developing an application. Proper memory management allows the application to run smoothly, prevent crashes, and provide an enjoyable experience for the users. Cloud-based platforms can use caching of memory that is essential for the application functions to optimize performance. Many options also allow for compression of files to save space. They typically provide memory monitoring tools that alert admins if there is a problem while also allocation more resources based on the needs of the application.
5. Distributed System and Networks: Draw It or Lost It needs to have the ability to communicate between various platforms as it looks to attract users who have different device and operating systems. When communicating between different platforms the development team must consider network connectivity, data security, scalability, and monitoring. The development team should consider using RESTful APIs. This helps ensure the data of the application users is secured while allowing communication across multiple platforms. Using redundant network links and balancing the load helps to ensure connectivity. This would allow users with any device to enjoy the application.
6. Security: Security is the number one priority for any application. Data encryption by using a tool such as SSL and RESTful APIs are a great place to start. Two factor authentication to verify user identities should be provided along with biometric options and strict password policies. The application itself should be designed with consideration for security. Data that is stored of a user should be kept to a minimum and users should be asked for their consent to store sensitive information about them. The development team should also allow minimal access to user information, only a need to basis. Lastly, there should be continuous monitoring for any suspicious activity and quick responses to any incidents detected.