

The Gaming Room

# **CS 230 Project Software Design**

Version 3.0

## 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 | 03/19/2023 | Nick Allen | Added exec summary, constraints, and domain model descriptions. |
| 2.0 | 03/28/2023 | Nick Allen | Added evaluation |
| 3.0 | 04/14/2023 | Nick Allen | Added recommendation |

**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)

Draw It or Lose It is a web-based game application based on the classic TV game, Win, Lose or Draw. Our aim is to create an entertaining game that lets teams compete against each other and the clock, trying to guess phrases, titles, or things from images that are drawn. We'll put together an easy-to-follow software design document and get the ball rolling on developing the game, making sure it meets all the requirements from The Gaming Room.

Our game app will let multiple teams with unique names join in, each having several players. We'll use unique identifiers for each game, team, or player, so there's only one instance of the game in memory at once. We're all about making a smooth and fun gaming experience, taking care of both software and hardware requirements as we build the game.

## 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)

* Adapting user interface and user experience (UI/UX):
  + Redesigning the game interface and interactions to suit web-based environments, ensuring ease of use and seamless navigation across various devices and screen sizes.
* Transition from native app to web technologies:
  + Migrating the game from native app technologies (e.g., iOS or Android) to web-based technologies (HTML5, CSS, JavaScript), potentially requiring a significant rewrite or refactoring of the original codebase.
* Browser limitations and compatibility:
  + Handling limitations imposed by different web browsers, such as rendering capabilities, performance differences, and unsupported features, to ensure a consistent gaming experience across popular browsers.
* Performance optimization for web-based games:
  + Addressing potential performance issues in a web-based environment, optimizing asset loading, network usage, and resource management to maintain smooth gameplay and responsiveness.

## [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 game application domain model consists of seven classes that collaborate and utilize object-oriented programming principles to provide an efficient and engaging gaming experience. The ProgramDriver class serves as the main entry point of the application, containing the main() method that runs and tests the game, showcasing the interaction between other classes.

SingletonTester is a utility class for testing the Singleton pattern in the GameService class. It has a testSingleton() method that ensures only one instance of GameService exists at any given time. The Entity class acts as the base class that holds common attributes like id and name, as well as common behaviors for Game, Team, and Player classes.

GameService is responsible for managing game instances, offering methods to add, retrieve, and count games. It employs the Singleton pattern to guarantee only one instance exists at any time, and it also keeps track of the next unique identifiers for games, teams, and players. The Game class signifies an individual game, with the capacity to add teams to it. Each game has a unique id and name, along with a list of teams. The addTeam() method enables adding teams to a game.

The Team class represents a single team in the game. Each team has a unique id, name, and a list of players. A team can add players using the addPlayer() method, which creates a new Player instance and adds it to the team. The Player class represents individual players within a team, each with a unique id and name. The Player class is derived from the Entity class, inheriting its attributes and behaviors.

All classes, including Game and Player, have toString() methods to represent their instances as human-readable strings, which helps in debugging and understanding the state of each object during the application's execution.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | macOS offers a stable and secure platform with strong web development support but has limited hardware options and higher upfront costs. It provides server-based deployment through macOS Server, but this is less common than other platforms. Licensing costs include a one-time fee for macOS Server software. | Linux is popular for servers due to its flexibility, stability, and low costs. It supports various server-based deployment methods, and distributions are typically free with optional commercial support. Linux may require a steeper learning curve for new users. | Windows offers a user-friendly environment with good web development support. It supports server-based deployment through Windows Server, but it can be less stable and secure compared to Linux. Licensing costs vary depending on edition and number of users or devices. | Mobile operating systems are not suitable for hosting web-based applications. They lack the necessary hardware and software infrastructure to support server-style configurations and large-scale web applications. |
| **Client Side** | For macOS web browser compatibility, implement a responsive, modern HTML interface. Perform cross-browser testing and optimization to manage rendering and performance disparities. Leverage popular frameworks (React, Angular, Vue.js) to expedite development and reduce platform-specific concerns. | To ensure compatibility with Linux web browsers, use a responsive, modern HTML interface. Employ cross-browser testing and optimization to handle rendering and performance differences. Utilize popular frameworks (React, Angular, Vue.js) to streamline development and minimize platform-specific issues. | To guarantee compatibility with Windows web browsers, develop a responsive, modern HTML interface. Conduct cross-browser testing and optimization to address rendering and performance variations. Utilize widely-used frameworks (React, Angular, Vue.js) to facilitate development and diminish platform-specific problems. | For both iOS and Android compatibility, create a responsive web design that adapts to different screen sizes and devices. Utilize progressive web app (PWA) technology or hybrid app frameworks like React Native or Flutter to ensure seamless performance across platforms. Conduct thorough testing and optimization to account for variations in devices, browsers, and operating system versions. This approach allows efficient development while maintaining consistent user experience and functionality. |
| **Development Tools** |  |  |  |  |

## Recommendations

1. **Operating Platform**: For the server platform of Draw It or Lose It, I recommend taking a cross-platform approach. This will allow The Gaming Room to easily expand the game to different computing environments. Consider using a cloud-based service such as Amazon Web Services (AWS), Microsoft Azure, or Google Cloud Platform. These platforms offer scalability, flexibility, and support for a variety of operating systems, including macOS, Linux, Windows, and mobile platforms such as iOS and Android.
2. **Operating Systems Architectures**: The chosen cloud-based platform will have the advantage of supporting multiple operating system architectures. AWS, for example, provides a wide range of virtual machines known as EC2 instances that can run on various operating systems such as Windows Server, macOS, and various Linux distributions. This means you can easily build and deploy your game server for multiple platforms while also taking advantage of the features and optimizations unique to each OS architecture.
3. **Storage Management**: To handle storage management, I suggest using a managed database service provided by the chosen cloud platform. For instance, Amazon RDS, Azure SQL Database, or Google Cloud SQL can be used. These services offer automatic backups, replication, and scaling, which will ensure the smooth operation of Draw It or Lose It. Additionally, using a managed database service will simplify maintenance tasks and improve the overall reliability of your storage solution.
4. **Memory Management**: The recommended cloud-based platform will handle memory management on the server-side for the Draw It or Lose It software. When you deploy your game server on a cloud platform, it will automatically allocate the necessary memory resources to ensure optimal performance. Additionally, you can easily scale your memory resources up or down based on the game's requirements. This will help maintain a responsive and efficient gaming experience, even as the number of players and platform demands increase.
5. **Distributed Systems and Networks**: To enable communication between various platforms, Draw It or Lose It can use RESTful APIs or GraphQL, which are widely supported on different operating systems and devices. These APIs will help establish a standard protocol for communication between the game server and clients. Furthermore, using a Content Delivery Network (CDN) can improve latency and reduce the impact of outages by caching game assets across various global locations. You should also consider implementing a monitoring and alert system to address connectivity and outage issues proactively.
6. **Security**: The recommended cloud-based platform offers various security features to protect user information on and between platforms. You can leverage encryption methods like SSL/TLS to secure data in transit, while using encryption at rest for stored data. Additionally, you can implement role-based access control (RBAC) to restrict access to sensitive data and services. It's also essential to follow best practices such as regular vulnerability scanning, patch management, and using multi-factor authentication (MFA) to further enhance the security of the Draw It or Lose It software.