

*Draw it or Lose it*

# **CS 230 Project Software Design Template**

Version 1.2

## 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 | 2/2/2025 | Caleb Baker | Added requirements, constraints, system view, domain model, evaluation, and recommendations. |
| 1.1 | 2/16/2025 | Caleb Baker | Updated previous parts based on feedback and added to evaluation. |
| 1.2 | 2/23/2025 | Caleb Baker | Updated recomendations.2 |

**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 Gamming Room is developing a new video game application named *Draw it or Lose it*, which is currently an android application, but they want to expand into a web-based distribution system. The game is similar to Pictionary, and has multiple teams and players doing timed drawing puzzles.

## Requirements

To meet the expectations of the client the program must include the following:

* Support multiple teams with multiple players.
* Support unique naming conventions for teams and players.
* Use a singleton design model to allow only one instance of the game service at a time.
* Use effective iterator patterns to manage the game.

## [Design Constraints](#_2et92p0)

The program is constrained by the following conditions:

* The game must be accessible across different web platforms.
* Data has to be constantly updated effectively to prevent conflicts.
* The application should be optimized for cloud/server-based deployment.
* Security will need to be implemented for a web-based environment.

## [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)

This UML class diagram demonstrates the relationships and purposes of this programs classes.

1. Entity: Serves as the base class for game elements (Game, Team, Player) and declares common variables id and name. Implements encapsulation to restrict direct access to its fields.
2. GameService: Implements the singleton pattern to maintain a single instance of the game service. This class is responsible for managing the lifecycle of Game instances and ensures unique ID assignment.
3. Game: Inherits from Entity using inheritance and contains a list of Team objects. Implements the composition principle by holding a list of Team objects.
   1. AddTeam: Creates a new Team instance and adds it to the game.
4. Team: Represents a team and contains a list of Player objects. Also follows composition as it holds Player instances.
   1. AddPlayer: Creates a player instance and adds it to team.
5. Player: Inherits from Entity and represents an individual player within a team.
6. ProgramDriver: Is the programs driver and coordinates game logic.
7. SingletonTester: Is used to initialize the singleton pattern of 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** | Creates a stable and secure environment but lacks the server hosting support. | Great stability, security, and is open source which can help with scalability and cost. | Great stability, security, but would be more expensive due to licensing. | Not suitable for hosting. |
| **Client Side** | Has great web-based access and large user base. | Has great web-based access and could be optimized for Linux users. Lower casual user base. | Has great web-based access and large user base. | Mobile devices need a unique responsive web design. This would have the biggest difference in time and recourses than the other platforms. Has a large user base. |
| **Development Tools** | Java, Swift, IntelliJ IDEA, Xcode, Eclipse, Gradle, Maven, Docker, JUnit, Selenium. | Java, Python, Bash, IntelliJ IDEA, Eclipse, VS Code, Gradle, Maven, Jenkins, JUnit, Selenium. | Java, C#, .NET, IntelliJ IDEA, Visual Studio, Eclipse, Gradle, MSBuild, Jenkins, JUnit, NUnit, Selenium. | Java (Android), Swift (iOS), Android Studio, Xcode, Gradle, Fastlane, Firebase, JUnit, Espresso, XCTest |

## 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 hosting, Linux is the best option since it’s stable, secure, and free to use. It also handles multiple processes well, making it ideal for running game servers. For development and testing, Windows and macOS should also be used since they match what most users will be playing on and have strong development tools.
2. **Operating Systems Architectures**: The game server will run on Linux with a monolithic kernel, which is great for performance because everything runs in the same space. The system will use process scheduling to handle multiple players smoothly. For easier updates and scaling, the game will be split into microservices using Docker and Kubernetes, so different parts of the game (like login, game logic, leaderboards) can be managed separately.
3. **Storage Management**: A relational database like MySQL or PostgreSQL will store player data, game stats, and team info. Caching (using Redis or Memcached) will speed up access times for frequently used data. Since the game needs to scale, storage will be managed using paging techniques to optimize memory use, and a distributed file system will be used to keep everything backed up and accessible.
4. **Memory Management:** Since the game is being developed in Java, memory management will be largely handled by automatic garbage collection, which will free up unused memory without manual intervention; However, additional paging and segmentation techniques will be implemented to optimize memory allocation for different game processes. Demand paging will ensure that only necessary data is loaded into memory, reducing overall RAM usage. In cases where memory becomes constrained, swapping will be used to temporarily move inactive game sessions to disk, ensuring that active players do not experience lag or performance issues.
5. **Distributed Systems and Networks**: The game will be built as a distributed system using a client-server model, where the core game logic runs on the server while clients communicate through RESTful APIs. To ensure smooth gameplay, load balancing using NGINX or HAProxy will be implemented to evenly distribute network traffic across multiple servers, preventing any single server from becoming overloaded. Redundancy and failover mechanisms will also be in place to ensure uptime even in the event of hardware failures.
6. **Security**: Security is a top priority, and multiple layers of protection will be implemented. Role-based access control (RBAC) will be used to limit user permissions and prevent unauthorized access. SSL/TLS encryption will be applied to secure data transmission, while AES encryption will be used to protect stored user information. Authentication will be handled using OAuth 2.0 or JWT tokens, ensuring that only legitimate users can access the game. To further enhance security, firewalls and DDoS protection will be in place to prevent cyberattacks, and regular security audits will be conducted to identify and fix potential vulnerabilities.