

The Gaming Room

# **CS 230 Project Software Design**

Version 3.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc152511734)

[Table of Contents 2](#_Toc152511735)

[Document Revision History 2](#_Toc152511736)

[Executive Summary 3](#_Toc152511737)

[Requirements 3](#_Toc152511738)

[Design Constraints 3](#_Toc152511739)

[System Architecture View 3](#_Toc152511740)

[Domain Model 3](#_Toc152511741)

[Evaluation 4](#_Toc152511742)

[Recommendations 7](#_Toc152511743)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | <11/19/23> | Thanh Nguyen | Initial version of Gaming Room  All the information that needs for the program to run |
| 2.0 | <12/03/23> | Thanh Nguyen | Update Evaluation of each platform |
| 3.0 | <12/09/23> | Thanh Nguyen | Update 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 aims to transition its Android game, Draw It or Lose It, into a web-based format to cater to multiple platforms. The challenge lies in efficiently streamlining the development process. This document outlines the proposed software design solution, including addressing unique team and game names, ensuring singular instances in memory, and incorporating software design patterns like the singleton and iterator patterns. The development of a robust, scalable, and user-friendly web-based application is critical for the success of Draw It or Lose It on various platforms.

## Requirements

The client's business and technical requirements include the ability for multiple teams with multiple players, unique team and game names, and singular instances in memory. These are vital for a seamless gaming experience on diverse platforms.

## [Design Constraints](#_2et92p0)

Designing the game application in a web-based distributed environment imposes constraints related to network latency, data synchronization, and platform compatibility. The implications involve careful consideration of data storage, communication protocols, and user interface design to ensure a consistent and responsive gaming experience across platforms.

## [System Architecture View](#_ilbxbyevv6b6)

## [Domain Model](#_8h2ehzxfam4o)

The UML class diagram illustrates the relationships between key entities in the game application: Game, Team, Player, and the base class Entity. Inheritance is employed, with Game inheriting from Entity. Object-oriented principles like encapsulation and abstraction are demonstrated in modeling common attributes and behaviors in the Entity class. The iterator pattern is utilized to ensure unique names for teams and games, enhancing the efficiency of operations.

**"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 provide a stable environment, but deployment costs may be higher. Consider macOS Server for hosting. | Linux servers are cost-effective and widely used for web hosting. Popular choices include Ubuntu Server or CentOS. | Windows Server is suitable for hosting applications developed using Microsoft technologies. Licensing costs should be considered. | For server-side interactions with mobile devices, implement backend services using technologies like Node.js, Django, or Spring Boot, ensuring compatibility with mobile communication protocols (e.g., RESTful APIs). |
| Client Side | Development for Mac clients may require expertise in Objective-C or Swift. | Linux clients may vary, and compatibility should be ensured for popular distributions. | Windows clients often use C# or C++ for application development. | Development for mobile devices requires knowledge of platform-specific languages like Kotlin or Swift. |
| Development Tools | Mac offers a stable environment for hosting web applications but may have higher development costs. It is well-suited for multimedia applications like games. | Linux is cost-effective and efficient for server-side hosting but may require more expertise for development. | Windows provides a user-friendly development environment but may have higher licensing costs. | Mobile devices have varying characteristics, and development should consider platform-specific features and constraints. |

**Evaluate the characteristics, advantages, and weaknesses of various platforms.**

1. **Server side**

|  |  |  |
| --- | --- | --- |
| **Operating Platform** | **Server-Based Deployment Method** | **Potential Licensing Costs** |
| Linux | Yes | Varies |
| Mac | Yes | Varies |
| Windows | Yes | Varies |

* All three traditional operating platforms (Linux, Mac, Windows) offer server-based deployment methods.
* Linux is an open-source platform, providing cost advantages for server deployment.
* Mac and Windows have varying commercial licensing costs, potentially impacting the client's budget.

1. **Client Side**

* Compatibility Across Platforms:

Desktop Clients (Linux, Mac, Windows): Utilize modern, responsive HTML interfaces for web browsers.

Mobile Platforms (iOS, Android): Ensure compatibility with web browsers on mobile devices.

* Development Process Requirements:

Cost: Consider additional development effort and testing to ensure cross-platform compatibility.

Time: Increased development time due to diverse platform requirements.

Expertise: Developers need skills in HTML, CSS, and JavaScript for web interfaces, along with expertise in mobile development for iOS and Android.

1. **Development Tools**

|  |  |  |  |
| --- | --- | --- | --- |
| Operating Platform | Programming Languages and Tools | Impact on Development Team | Licensing Costs |
| Linux | HTML, CSS, JavaScript | Collaboration may require specialized knowledge | Open Source |
| Mac | HTML, CSS, JavaScript | Collaboration may require specialized knowledge | Commercial |
| Windows | HTML, CSS, JavaScript | Collaboration may require specialized knowledge | Commercial |
| Mobile Platforms | HTML, CSS, JavaScript, Mobile SDKs (iOS, Android) | Separate expertise may be required for mobile development | Varies |

## Recommendations

1. **Operating Platform:**

* We suggest using cloud services like AWS or Google Cloud for flexibility and scalability.
* Consider AWS Lambda, a serverless computing option, to scale automatically based on demand and reduce infrastructure management workload.
* Explore Google Kubernetes Engine (GKE) for efficient deployment, scaling, and management of containerized applications.

1. **Operating Systems Architectures:**

* Opt for a microservices approach for scalability and easier maintenance.
* Use Docker containers to package microservices consistently across different environments.
* Implement an API Gateway to manage communication between microservices,

1. **Storage Management:**

* Choose a cloud-based database like Amazon DynamoDB for efficient and scalable storage.
* Leverage DynamoDB's NoSQL capabilities for flexible data access.
* Consider the serverless option for DynamoDB to eliminate the need for capacity planning.
* Implement data caching mechanisms, such as Amazon ElastiCache, to boost performance.

1. **Memory Management:**

* Use programming languages like Python or Java with strong garbage collection mechanisms.
* Optimize memory usage and application performance by implementing efficient data structures and algorithms.

1. **Distributed Systems and Networks:**

* Set up RESTful APIs for communication between platforms, keeping in mind potential network outages.
* Document and standardize RESTful API interfaces using tools like Swagger or OpenAPI.
* Implement a message queue system, such as Apache Kafka, for asynchronous communication between different parts of the system.

1. **Security:**

* Ensure secure communication using HTTPS.
* Use OAuth 2.0 for user authentication and authorization.
* Conduct regular security audits using tools like OWASP ZAP or Nessus to identify and fix potential vulnerabilities.
* Consider implementing end-to-end encryption for sensitive data transmitted between clients and servers to enhance data protection.