

Draw It or Lose It

# **CS 230 Project Software Design**

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 | 02/24/2024 | Jeffrey Conwi | Initial creation of the software design document for "Draw It or Lose It" web application development. |

## [Executive Summary](#_sbfa50wo7nsh)

Creative Technology Solutions (CTS) presents a comprehensive software design document aimed at developing a web-based version of The Gaming Room's "Draw It or Lose It". The transition from an Android-only app to a cross-platform web application addresses the client's need for broader accessibility and user engagement. The proposed solution leverages modern web technologies to ensure a seamless, interactive gaming experience across various devices. Critical considerations include resource allocation, timeline estimation, and risk management strategies to navigate potential technical challenges.

## Requirements

*This section would list the specific business and technical requirements as communicated by The Gaming Room, such as the need for cross-platform compatibility, unique team and game naming conventions, and ensuring only one instance of the game exists in memory at any time.*

## [Design Constraints](#_2et92p0)

Developing "Draw It or Lose It" for a web-based distributed environment introduces constraints such as browser compatibility, network latency, and the limitations of web technologies in replicating an Android app's functionalities. These constraints necessitate a design approach that prioritizes responsiveness, efficient data handling, and scalability to accommodate a broad user base.

## [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 for "Draw It or Lose It" outlines the structure and relationships between several key classes involved in the application. These classes include ProgramDriver, SingletonTester, Entity, GameService, Game, Team, and Player.

ProgramDriver Class: Serves as the entry point of the application with a main method (+main()). It uses (<<uses>>) the SingletonTester class to test singleton behavior within the application.

SingletonTester Class: Contains a method (+testSingleton()) to test the singleton pattern implementation in the GameService class, ensuring that only one instance of GameService can exist at a time.

Entity Class: Acts as a base class for Game, Team, and Player classes. It defines common attributes such as id (of type long) and name (of type String), along with constructors and methods

(+Entity(id:long, name:String), +getId(): long, +getName(): String, +toString(): String) for common operations.

GameService Class: Implements the singleton pattern to ensure a single instance. It manages games and their IDs, player IDs, and team IDs through a collection of Game objects and static attributes for tracking next IDs. It provides methods for adding and retrieving games, getting game counts, and generating next IDs for players and teams

(+getInstance(): GameService, +addGame(name:String): Game, +getGame(id:long): Game, +getGame(name:String): Game, +getGameCount(): int, +getNextPlayerId(): long, +getNextTeamId(): long).

Game Class: Represents a game consisting of multiple teams. It contains a list of Team objects and methods for adding teams and converting game information to a string format (+Game(id: long, name: String), +addTeam(name:String): Team, +toString(): String).

Team Class: Represents a team consisting of multiple players. It holds a list of Player objects and provides methods for adding players and representing team information as a string (+Team(id: long, name: String), +addPlayer(name:String): Player, +toString(): String).

Player Class: Represents a player with methods for initializing player instances and converting player information to a string (+Player(id: long, name: String), +toString(): String).

The relationships between classes are indicated by lines and arrows. GameService is associated with Game, Game is associated with Team, and Team is associated with Player, each with a "0...\*" multiplicity indicating a one-to-many relationship. Furthermore, Game, Team, and Player classes are all derived from the Entity class, indicated by a line with an open arrow pointing towards Entity, demonstrating an inheritance relationship.

Object-Oriented Programming Principles Demonstrated:

Inheritance: Utilized by the Game, Team, and Player classes which inherit from the Entity class, sharing common attributes and methods.

Encapsulation: Demonstrated through the use of private attributes and public methods to safely access and modify the data.

Singleton Pattern: Implemented by the GameService class to ensure that only one instance of this class can exist at any time, controlling game state management across the application.

**"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** | **Advantages**: Stable, UNIX-based, robust security, versatile development environment.  **Weaknesses**: Higher hardware/software costs, smaller user base limits scalability. | **Advantages**: Stability, security, open-source, cost-effective, highly customizable, scalable.  **Weaknesses**: Requires technical expertise, challenges with specific distributions. | **Advantages**: Easy-to-use interface, integration with Microsoft products, .NET support.  **Weaknesses**: Higher licensing costs, security concerns, performance issues. | **Advantages**: Not applicable for server-side hosting  **Weaknesses**: Impractical due to resource limitations and security concerns |
| **Client Side** | More expensive due to hardware and IDE requirements; good support for modern web technologies across browsers. | Diverse distributions increase compatibility testing; benefits from open-source cost efficiency. | Large user base requires broad compatibility; diverse tools range from open source to commercial. | Developing for mobile devices requires considering diverse screen sizes, operating systems (iOS, Android), and performance characteristics. Tools like React Native or Flutter can streamline development for multiple mobile platforms but require specific expertise. |
| **Development Tools** | Tools: Xcode, Visual Studio Code, JetBrains IDEs. Languages: Swift, Objective-C, JavaScript, Python. | Tools: Vim, Emacs, Eclipse, IntelliJ IDEA, Visual Studio Code Languages: Python, JavaScript, Ruby, PHP. | Tools: Visual Studio Code, JetBrains Rider.  Languages: C#, VB.NET, JavaScript, Python. | Tools: Android Studio, Xcode, Flutter, React Languages: Java, Kotlin, Swift, Dart. |

## 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 server-side operations, Linux is highly recommended for hosting "Draw It or Lose It". Linux offers robustness, scalability, and cost-effectiveness, making it an ideal choice for managing the server demands of a popular web-based game. Linux servers are known for their stability and security, which are crucial for real-time, multiplayer gaming environments.
2. **Operating Systems Architectures**: Linux operates under a monolithic kernel architecture, which means that the kernel (core of the operating system) provides a wide range of functionalities within a single large process. This architecture facilitates efficient interaction between hardware and software components, crucial for the performance-intensive demands of "Draw It or Lose It". Additionally, the Linux system can be tailored to optimize resource usage, which is essential for handling high numbers of simultaneous game sessions.
3. **Storage Management**: Considering the data demands of "Draw It or Lose It", which includes player data, game state management, and real-time game statistics, a distributed file system like GlusterFS or a block storage solution like Ceph would be appropriate. These systems are scalable and offer redundancy and high availability, which are necessary for ensuring that game data is consistently accessible and secure.
4. **Memory Management**: Linux uses several memory management techniques to optimize the performance of applications:  
   - Virtual Memory: Using disk space as an extension of RAM, allowing more applications to run simultaneously.  
   - Swapping: Temporarily transferring inactive portions of memory to disk, making room for active processes in RAM.  
   - Page Caching: Storing page files of frequently accessed data to reduce read-access times.  
   These techniques collectively ensure that "Draw It or Lose It" can handle multiple instances efficiently without degrading performance
5. **Distributed Systems and Networks**: "Distributed It or Lose It" can be achieved by implementing a client-server architecture where the server handles game logic, state management, and data storage, while clients (players' devices) are responsible for presenting the user interface and inputs. Utilizing WebSockets or TCP/IP for real-time communication ensures minimal latency and real-time responsiveness. Consider the network topology and implement redundancy and failover strategies to handle potential network outages and ensure seamless gameplay.
6. **Security**: To protect user information and ensure secure data transactions between various platforms, implement the following security measures: SSL/TLS for encrypting data in transit, OAuth for secure, token-based user authentication, HTTPS for secure communication over the network, and Regular security audits and updates to address vulnerabilities.
7. These recommendations are designed to optimize the expansion of "Draw It or Lose It" into a robust, scalable, and secure web-based gaming environment that supports multiple platforms and handles large volumes of concurrent users.