
Draw It or Lose It Web Application
**CS 230 Project Software Design Template**
Version 1.0

## Table of Contents



### Document Revision History

| Version | Date | Author | Comments |
|---|---|---|---|
| 1.0 | 05/24/2025 | Nicholas Harris | Initial software design document for Draw It or Lose It web application |
| 3.0 | 06/22/2025 | Nicholas Harris | Added Recommended section (talking about enterprise level hosting system) |

## 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

The Gaming Room has contracted Creative Technology Solutions (CTS) to develop a web-based version of their existing Android game "Draw It or Lose It." This document outlines the software design approach for creating a distributed, multi-platform gaming application that maintains the core gameplay while expanding accessibility across different operating systems and devices.

**Current State:** Draw It or Lose It exists only as an Android application with limited platform reach and single-device gameplay.

**Proposed Solution:** A web-based application utilizing object-oriented design patterns to ensure scalability, maintainability, and cross-platform compatibility. The solution will implement singleton and iterator design patterns to manage game instances and ensure unique naming conventions while supporting multiple concurrent teams and players.

**Critical Success Factors:**

- Single game instance management to prevent memory conflicts
- Unique identification system for games, teams, and players
- Cross-platform web deployment capability
- Scalable architecture supporting multiple simultaneous games
- Efficient resource management for image rendering and game state tracking

This approach will allow The Gaming Room to expand their market reach while maintaining game integrity and performance across diverse computing environments.

## Requirements

< Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client's business and technical requirements in a clear and concise manner.>*

### Design Constraints

**Web-Based Distributed Environment Constraints:**
**Network Latency and Bandwidth:**

- Real-time image rendering requires optimized data transmission
- 30-second progressive image reveal demands consistent network performance
- Multiple concurrent users create bandwidth competition
- *Implication:* Requires efficient image compression and caching strategies

**Browser Compatibility:**

- Different browsers handle JavaScript and rendering differently
- HTML5 canvas support variations across platforms
- WebSocket implementation differences for real-time communication
- *Implication:* Extensive cross-browser testing and fallback mechanisms required

**State Management:**

- Game state must persist across network interruptions
- Player sessions need reliable reconnection capabilities
- Team coordination requires synchronized state updates
- *Implication:* Robust session management and data persistence implementation necessary

**Security Constraints:**

- Client-side code exposure in web browsers
- Cross-site scripting (XSS) vulnerability prevention
- User authentication and session security
- *Implication:* Server-side validation and secure communication protocols essential

**Scalability Limitations:**

- Single game instance constraint conflicts with multi-user web environment
- Memory management across distributed server instances
- Load balancing complexity with singleton pattern requirements
- *Implication:* Careful architecture planning for horizontal scaling while maintaining singleton integrity

**Performance Constraints:**

- JavaScript execution limitations in browsers
- Mobile device processing power variations
- Progressive image loading performance requirements
- *Implication:* Optimized code structure and resource management critical for consistent user experience

## System Architecture View

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

**UML Class Diagram Analysis:**

The provided UML diagram demonstrates a well-structured object-oriented design implementing several key design patterns and principles within the com.gamingroom package:

**Class Hierarchy and Relationships:**

**Entity (Base Class):**

- Serves as the foundation for all game objects with common attributes: id (long) and name (String)
- Provides comprehensive method suite: default Entity() constructor, parameterized Entity(id: long, name: String) constructor
- Accessor methods: getId(): long, getName(): String, and toString(): String
- Demonstrates **abstraction** by providing a unified interface for all game entities

**Game Class:**

- Inherits from Entity, gaining unique identification and naming capabilities
- Contains teams: List<Team> attribute showing **composition** relationship with Team objects
- Implements key methods: Game(id: long, name: String) constructor, addTeam(name: String): Team, and toString(): String
- Demonstrates **encapsulation** of team management logic through controlled team creation

**Team Class:**

- Extends Entity maintaining consistent identification across the application
- Contains players: List<Player> attribute showing **aggregation** with Player objects
- Provides Team(id: long, name: String) constructor, addPlayer(name: String): Player, and toString(): String
- Demonstrates **single responsibility principle** by managing only player-related operations

**Player Class:**

- Cleanest implementation extending Entity base class
- Simple constructor Player(id: long, name: String) and toString(): String method
- Represents individual participants with minimal complexity

**GameService Class (Singleton Implementation):**

- Implements **Singleton Pattern** through comprehensive static management system
- Static attributes: games: List<Game>, nextGameId: long, nextTeamId: long, nextPlayerId: long, service: GameService
- Core singleton methods: GameService() private constructor, getInstance(): GameService
- Business logic methods: addGame(name: String): Game, getGame(id: long): Game, getGame(name: String): Game
- ID generation utilities: getNextGameId(): long, getNextPlayerId(): long, getNextTeamId(): long

**Supporting Classes:**

- **ProgramDriver:** Contains main() method for application entry point and testing

- **SingletonTester:** Provides testSingleton() method to validate singleton pattern implementation

**Object-Oriented Principles Demonstrated:**

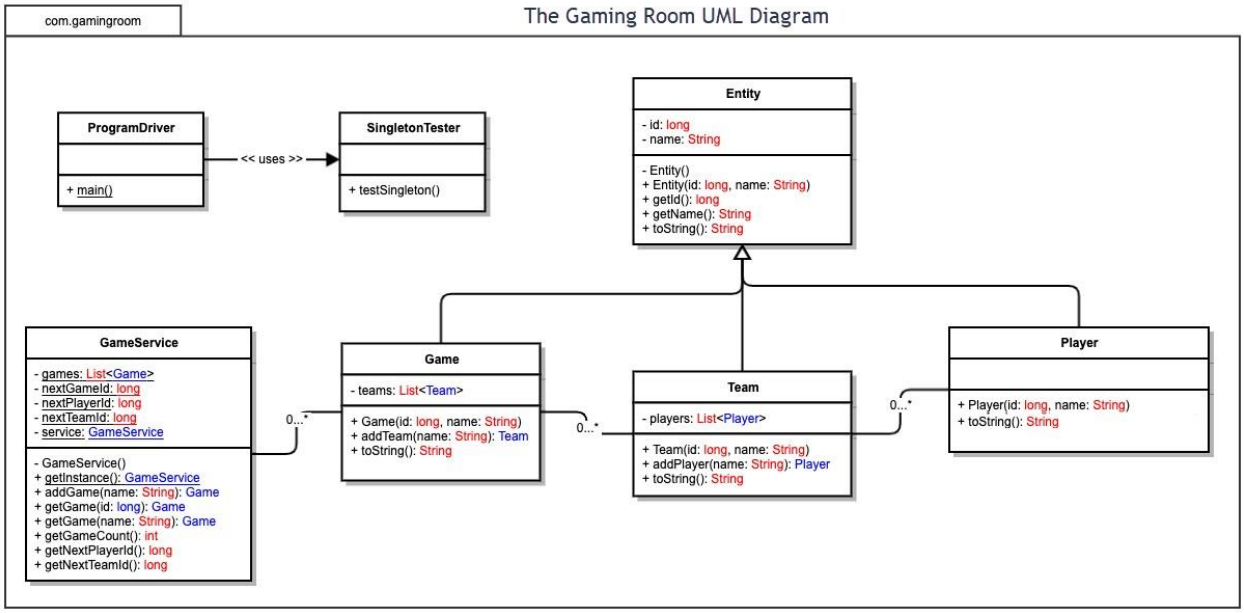
1. **Inheritance:** Game, Team, and Player classes all extend Entity base class, promoting code reuse and maintaining consistent id and name attributes across all game objects
2. **Encapsulation:** Each class manages its own data through private/protected attributes and provides controlled access via public methods like addTeam(), addPlayer(), and getInstance()
3. **Polymorphism:** Entity references can point to Game, Team, or Player objects, enabling flexible object handling through common interface methods
4. **Abstraction:** Entity class abstracts common properties (id, name) and behaviors (toString(), constructors) shared by all game objects

**Design Pattern Implementation:**

1. **Singleton Pattern:** GameService implements classic singleton with:
    a. Private constructor preventing external instantiation
    b. Static service attribute holding single instance
    c. Public getInstance() method providing controlled access
    d. Static ID generators ensuring unique identifiers across entire application
2. **Iterator Pattern:** Implemented within GameService methods (getGame(name: String), addGame()) and anticipated in Team's addPlayer() and Game's addTeam() methods for name validation and collection traversal

**Software Requirements Fulfillment:**

- **Multiple Teams per Game:** Achieved through Game class's teams: List<Team> composition and addTeam() method
- **Multiple Players per Team:** Implemented via Team class's players: List<Player> aggregation and addPlayer() method
- **Unique Naming:** Iterator pattern implementation in service methods enables name validation before object creation
- **Single Game Instance:** Singleton pattern in GameService ensures centralized game management with only one service instance
- **Unique Identifiers:** Static counters (nextGameId, nextTeamId, nextPlayerId) in GameService provide unique IDs for all entities




## Evaluation

Based on the comprehensive platform analysis, Linux emerges as the optimal server-side platform due to its superior stability, cost-effectiveness, and robust security features essential for web-based gaming applications. For client-side development, a responsive web design approach ensures compatibility across all platforms while minimizing development complexity. The evaluation reveals that cross-platform web technologies provide the most efficient path to market, allowing The Gaming Room to reach users on Mac, Linux, Windows, and mobile devices simultaneously through a single codebase, significantly reducing development time and maintenance costs compared to platform-specific native applications.

| Development Requirements | Mac | Linux | Windows | Mobile Devices |
|---|---|---|---|---|
| **Server Side** | Mac servers offer excellent performance and reliability but come with higher licensing costs and limited scalability options. macOS Server provides robust security features and seamless integration with Apple ecosystems. However, the proprietary nature and premium pricing make it less cost-effective for large-scale web deployments compared to open-source alternatives. | Linux provides superior stability, security, and cost-effectiveness for web server hosting. Open-source nature eliminates licensing fees while offering extensive customization options. Supports high-performance web servers like Apache and Nginx with excellent scalability. Large community support ensures rapid security updates and troubleshooting resources. | Windows Server offers enterprise-grade hosting capabilities with strong .NET framework integration and IIS web server support. Provides excellent scalability and robust security features with Active Directory integration. However, licensing costs are significantly higher than open-source alternatives, with per-core licensing fees that can become expensive for large deployments. Strong Microsoft ecosystem integration benefits organizations already using Windows infrastructure. | Mobile devices are not suitable for server-side hosting due to limited processing power, storage capacity, and network reliability. Battery life constraints and intermittent connectivity make mobile platforms inappropriate for continuous server operations requiring 24/7 availability and consistent performance. |

| | | | | |
|---|---|---|---|---|
| **Client Side** | Mac clients provide excellent user experience with high-quality displays and intuitive interfaces. Strong web browser support with Safari optimization. Development costs may be higher due to testing requirements across different macOS versions. Limited market share compared to Windows but represents valuable demographic for gaming applications. | Linux clients offer flexibility and customization options appealing to technical users. Multiple browser options with strong open-source support. Lower development costs due to open standards compliance. Smaller desktop market share but growing popularity among developers and technical professionals who may be early adopters. | Windows represents the largest desktop market share, making it essential for broad user reach. Excellent browser compatibility with Chrome, Firefox, and Edge. Familiar interface reduces user training requirements. Development considerations include supporting various Windows versions and ensuring consistent performance across different hardware configurations. | Mobile devices represent the fastest-growing gaming platform with touch-based interfaces requiring specialized user experience design. Responsive web design essential for supporting various screen sizes and orientations. Performance optimization critical due to limited processing power and battery constraints. Cross-platform compatibility needed for iOS and Android browsers. |

| | | | | |
|---|---|---|---|---|
| **Development Tools** | Mac development benefits from Xcode and excellent UNIX-based development environment. Strong support for web development frameworks and debugging tools. Higher hardware costs but superior build quality. Limited to Apple hardware, potentially increasing development team equipment expenses. | Linux offers comprehensive open-source development tools including robust IDEs, compilers, and testing frameworks. Cost-effective development environment with excellent performance. Strong community support and extensive documentation. Familiar environment for server-side development teams. | Windows provides Visual Studio and comprehensive Microsoft development ecosystem. Strong integration with Azure cloud services and .NET frameworks. Familiar development environment for many programmers. Enterprise-grade debugging and profiling tools available. Professional development tools like Visual Studio Professional require licensing fees that should be factored into development costs. | Mobile development requires specialized tools including device emulators, responsive design testing frameworks, and performance monitoring utilities. Cross-platform testing essential due to device fragmentation. Higher complexity for ensuring consistent experience across different mobile browsers and operating systems. |

## Recommendations

Based on the analysis of various systems architectures and The Gaming Room's requirements for expanding Draw It or Lose It to multiple computing environments, the following recommendations provide a practical roadmap for successful cross-platform deployment.

### Operating Platform

**Recommendation:** Linux-based server architecture using Ubuntu Server LTS as the primary operating platform.

Linux emerges as the optimal choice for The Gaming Room's server infrastructure due to its cost-effectiveness, stability, and security advantages. The open-source nature eliminates licensing costs that would be required with Windows Server, while providing superior uptime and reliability compared to other platforms. Linux's robust security model and rapid patch deployment through the open-source community make it ideal for protecting user data and maintaining game integrity. The platform's efficient resource management ensures maximum CPU and memory allocation to the gaming application rather than overhead processes, directly supporting more concurrent users with better response times.

### Operating Systems Architectures

**Recommendation:** Containerized microservices architecture using Docker containers with load balancing capabilities.

The recommended architecture implements a modular approach where different application components operate as independent services. The Game Service manages game instances, Team Service handles team operations, Player Service manages authentication, and Image Service controls progressive image reveals. Each service runs in Docker containers, allowing independent scaling based on demand. Nginx serves as a reverse proxy and load balancer, distributing requests across multiple application instances while providing SSL termination. This architecture supports the singleton pattern requirements of GameService while enabling horizontal scaling through container orchestration. Health monitoring ensures failed containers are automatically replaced, maintaining system availability during peak usage periods.

### Storage Management

**Recommendation:** Hybrid storage approach combining PostgreSQL for persistent data with Redis for session management and cloud storage for static assets.

PostgreSQL serves as the primary database for all persistent game data including user profiles, game configurations, and team compositions. Its ACID compliance ensures data consistency crucial for maintaining game integrity during concurrent operations. The database schema includes proper indexing on frequently queried fields to support unique naming requirements efficiently. Redis provides high-performance in-memory storage for active game sessions, player states, and frequently accessed data. Game session information persists in Redis with automatic cleanup of completed games. Static assets like game images utilize cloud-based object storage with Content Delivery Network integration for optimal loading performance across different

geographic locations. Automated backup strategies ensure data recovery capabilities for both PostgreSQL and Redis systems.

**Memory Management**

**Recommendation:** Optimized Java Virtual Machine configuration with garbage collection tuning and connection pooling strategies.

The JVM configuration utilizes G1 garbage collection optimized for low-latency gaming applications with heap sizes configured appropriately for concurrent game sessions. Key parameters include targeting pause times under 100ms to maintain responsive gameplay during garbage collection cycles. The application implements object pooling for frequently created game objects to reduce memory allocation overhead. Database connection pooling through HikariCP maintains optimal connection counts to prevent resource waste while ensuring availability during peak usage. A multi-level caching strategy manages image assets efficiently, with application-level caching for frequently accessed images and Redis caching for shared storage across instances. Memory monitoring provides real-time visibility into utilization patterns with automated alerting for proactive scaling when thresholds are exceeded.

**Distributed Systems and Networks**

**Recommendation:** Multi-server deployment with WebSocket communication for real-time synchronization and message queuing for reliable inter-service communication.

The distributed architecture deploys application instances across multiple servers to ensure scalability and redundancy. WebSocket connections enable real-time communication between clients and servers for live game updates, including image progression and team coordination. The implementation includes automatic reconnection capabilities to handle network interruptions gracefully, with fallback mechanisms for clients with connectivity restrictions. Apache Kafka manages inter-service communication and event distribution, organizing events by type with appropriate processing strategies. An API Gateway provides external clients with unified access to internal services while implementing rate limiting and authentication. Content Delivery Network integration distributes static assets globally while providing DDoS protection. The system includes failover capabilities that automatically redirect traffic to healthy servers when issues occur, ensuring consistent game state synchronization across different client platforms.

**Security**

**Recommendation:** Multi-layered security implementation using OAuth 2.0 authentication, TLS encryption, and comprehensive input validation with advanced threat protection.

OAuth 2.0 with OpenID Connect provides secure user authentication supporting multiple identity providers while JSON Web Tokens manage session state with appropriate expiration policies. Role-based access control ensures proper permissions for different user types while integrating with the GameService singleton pattern for game-specific authorization. All data transmission utilizes TLS 1.3 encryption with HTTP Strict Transport Security headers to prevent downgrade attacks. Database encryption at rest protects stored information using industry-standard algorithms. Comprehensive input validation at multiple application layers prevents SQL

injection and Cross-Site Scripting attacks through parameterized queries and Content Security Policy headers. A Web Application Firewall provides defense against common attacks while including gaming-specific rules to prevent cheating attempts. Rate limiting protects against brute force attacks and ensures fair resource allocation. Session management implements secure cookie configurations with automatic timeout and secure invalidation procedures. Security monitoring through logging systems provides real-time threat detection with automated alerting capabilities. The security framework includes regular vulnerability assessments and incident response procedures to maintain protection against emerging threats across all supported platforms.