The Gaming Room

CS 230 Project Software Design Template

Version 1.0

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	2/21/2026	Romane Dorrel	Project Three Submission

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

Creative Technology Solutions has been hired by The Gaming Room, the company behind the Android game Draw It or Lose It, which is based on the 1980s television game Win, Lose or Draw. The Gaming Room is attempting to expand their game beyond their current Android-only capabilities to a web-based game playable on multiple platforms. The expanded version should maintain current key game aspects such as allowing for multiple teams, timed rounds, progressive image rendering, and a shortened guessing window for non-guessing teams.

To meet this requirement, the proposed solution is to have a centralized game management service to control the creation and coordination of games, teams, and players. Each game supports one or more teams, with each team composed of multiple players. The system enforces unique game and team names to prevent conflicts and improve the user experience during setup. A singleton-based service layer ensures that only one instance of the game management service exists in memory at any given time, allowing for consistent state management and unique identifier assignment across the application.

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

Distributed, Web-Based Environment

Rationale: The application must run in a web browser so it can be used on multiple devices and operating systems. This constraint requires thoughtful design to ensure the game adapts smoothly to different platforms, screen sizes, and user environments, providing a consistent experience for all players.

Multiple Players per Team

Rationale: Each team must support multiple players, which requires the system to properly group players under the correct team during gameplay. This constraint ensures that team-based collaboration functions as intended while also requiring the application to accurately track player participation, turns, and interactions within each team throughout the game.

Unique Naming and Centralized State Management

Rationale: The requirement for unique game and team names affects how games are created and tracked within the system. The application must check whether a name is already in use before allowing a new game or team to be created. To do this reliably, all name checks must be handled in one central place so that players receive accurate feedback when choosing names and duplicate games or teams are avoided.

Single Instance of Game Management Service

Rationale: Only one game management service can exist at a time to ensure the game runs consistently for all players. This single service is responsible for creating games, teams, and players and for keeping track of the overall game state. While this approach simplifies coordination and reduces errors, it also means the service must be carefully designed to handle multiple players and games at the same time without slowing down the experience.

Flexible Game Structure

Rationale: The game must support flexibility in how teams are organized, allowing for both single-team games and games with multiple competing teams. Each game can include one or more teams, and each team will include multiple players. This structure allows the game to accommodate different play styles while still maintaining clear relationships between games, teams, and players. The system must consistently track these relationships throughout the game lifecycle to ensure accurate gameplay and scoring.

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

The Entity class serves as a base class for Game, Team, and Player. It holds shared attributes such as a unique identifier and a name, which are common across all entities in the system. By introducing this base class, the design avoids duplication and ensures consistent handling of shared data.

The Player class represents an individual participant in the game. Players are associated with a specific team and are required for creating teams.

The Team class represents a group of players competing together within a game. Each team can have multiple players assigned to it, which is modeled as a zero-to-many relationship. A team is required to play the game.

The Game class represents a single game session. Each game can contain one or more teams, supporting both single-team and multi-team gameplay. This relationship is shown as a zero-to-many association, indicating that a single game can have any number of teams.

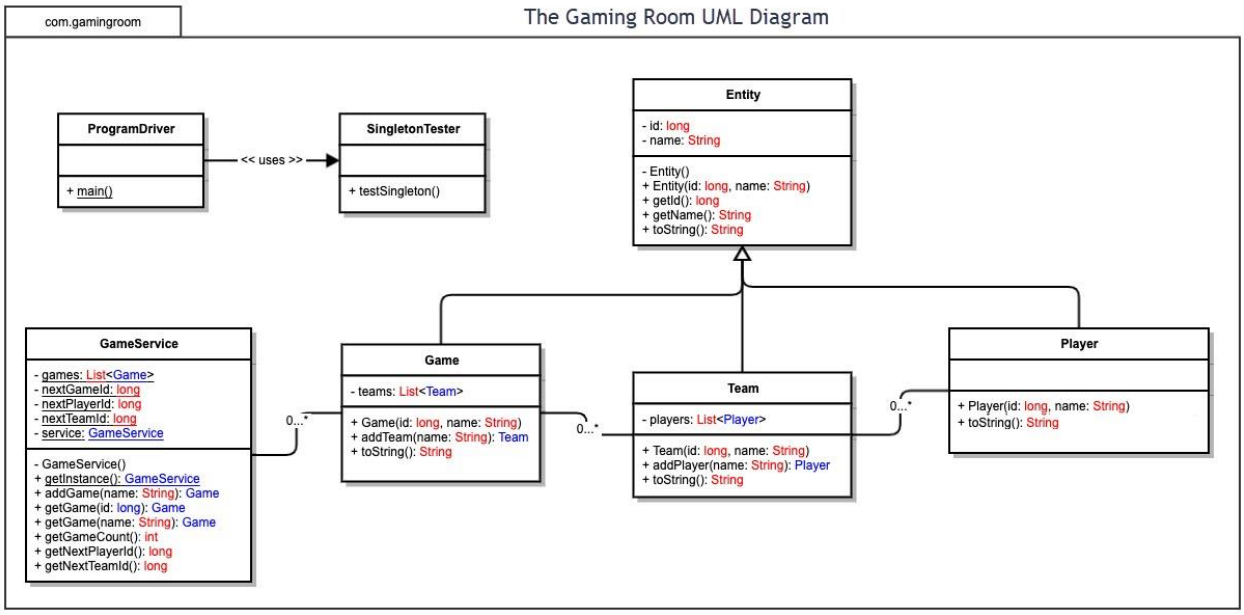
The GameService class appears to act as the controller of the game, managing and keeping track of all active game instances. It is also responsible for generating and maintaining unique identifiers for games, teams, and players, which helps prevent duplication and ensures consistent tracking throughout the application.

The ProgramDriver contains the main program responsible for running the application. It uses the SingletonTester to verify that only one instance of the game service is active at any given time.

Inheritance is used through the Entity base class, allowing Game, Team, and Player to share common attributes and behaviors while maintaining their distinct roles within the system.

Encapsulation is demonstrated by keeping entity data private and providing controlled access through methods, ensuring that internal state cannot be modified incorrectly.

Composition is used to represent the relationships between games, teams, and players, where teams exist within games and players exist within teams.



Evaluation

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	macOS does not provide a designated server-based deployment method for hosting large-scale web applications. The platform is primarily focused on end users and developers. Users would access and utilize the game through a web browser rather than hosting it locally. There are no licensing costs associated with accessing the game via a web browser on macOS.	Linux is an open-source operating system commonly used for hosting web applications. It is frequently used as the backend platform for web services accessed by both desktop and mobile devices. Linux offers strong scalability and stability for large deployments, with no operating system licensing costs, making it a cost-effective option for hosting a web-based game.	Windows offers a commercial operating system with dedicated server editions designed for hosting web applications. Windows Server is commonly used in enterprise and corporate environments. While it is capable of hosting the game, licensing costs can increase significantly as the deployment scales to support a large number of concurrent players.	Android and iOS devices allow flexible access to web-based applications through mobile browsers, enabling users to access and utilize the game on their mobile platforms. However, these operating systems do not provide the capability to host a production web application. There are no additional licensing costs associated with accessing the game through web browsers on Android or iOS devices.

Client Side	Supporting macOS as a client platform involves minimal development cost, as the game is accessed through a web browser rather than a native application. Development time remains low because no macOS-specific version is required, though some additional time is needed for browser testing, particularly with Safari. Standard web development expertise is sufficient, with limited platform-specific knowledge required.	Supporting Linux as a client platform involves minimal development cost, as users access the game through standard web browsers. Development time remains low because no Linux-specific version is required, though additional testing may be needed to account for different distributions, such as Ubuntu or Linux Mint, and browsers like Brave or Chromium. Standard web development expertise is sufficient, with little platform-specific knowledge required.	Supporting Windows as a client platform involves minimal development cost, as users access the game through standard web browsers. Development time remains low because no Windows-specific version is required, though testing across common browsers such as Chrome, Edge, and Firefox is necessary. Standard web development expertise is sufficient, with no specialized Windows-specific knowledge required.	Supporting mobile devices as client platforms involves low development cost, as the game is accessed through mobile web browsers rather than native applications. Development time may increase slightly due to the need for responsive design and touch-based input support. Web development expertise is sufficient, though familiarity with mobile browser behavior and performance considerations is beneficial.
Development Tool	The web-based application can be developed using standard programming languages regardless of the platform hosting or accessing the game. Front-end development uses HTML, CSS, and JavaScript, which are free and supported across all modern browsers and platforms. Backend development can be handled using Java or JavaScript in combination with SQL to manage game logic and data storage, also without licensing costs. Development can be completed using free IDEs such as Visual Studio Code or Eclipse, resulting in no additional licensing costs for development tools.			

Development Tools	macOS does not provide a viable option for hosting the game in a production server environment, so no additional deployment costs are associated with the macOS platform. macOS may be used for development if preferred, but it does not require a separate development effort, as the same programming languages and IDEs are available across platforms at no cost.	Deploying the application on Linux does not require operating system licensing fees and is one of the most commonly used open-source options for hosting web-based applications. Linux typically utilizes web servers such as Apache or Nginx for deployment, which also incur no additional licensing costs. Because development does not require platform-specific changes and deployment is only needed on a single platform such as Linux, multiple development teams are not necessary.	While Windows Server could be used for deployment, it would introduce additional licensing costs without providing significant benefits. Since the game is intended to be accessed across all platforms through a web browser and can be hosted effectively at no cost using a Linux server, the chosen technical requirements have minimal impact on the development team and do not justify the added expense.	Mobile platforms do not require changes to the core development or deployment approach. The primary considerations for mobile access involve ensuring responsive page layouts and optimized screen sizing to support different devices. All game logic and hosting remain centralized on the backend server. These requirements can be met using the same free development tools and programming languages, with no additional licensing costs.

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**:

   Linux Server is the recommended operating platform for hosting The Gaming Room's web-based application. Linux provides cost efficiency, high performance, and operational stability in server environments (The Linux Foundation, n.d.). For a multiplayer platform such as Draw It or Lose It, Linux supports scalability, efficient resource utilization, and reliable network communication. Its resource management capabilities support multiple concurrent game

sessions (The Linux Kernel Documentation, n.d.). As an open-source platform, Linux also eliminates licensing costs and integrates effectively with cloud and containerized deployments.

2. **Operating Systems Architectures**:

The Linux operating platform follows a layered architectural model consisting of hardware, the kernel, system libraries, the shell, and user-space applications. At its core, the Linux kernel manages communication between software and hardware and performs essential functions such as process scheduling, memory management, file system control, device driver coordination, and network communication. Linux utilizes a monolithic kernel design in which core operating system services execute within kernel space for efficiency and performance.

The separation between kernel space and user space enhances system stability and security by isolating application processes from core system operations. Linux also supports multitasking, multiuser access control, and a fully integrated networking stack, making it well suited for distributed and concurrent web-based applications (The Linux Kernel Documentation, n.d.).

3. **Storage Management**:

Amazon Web Services (AWS) is the recommended cloud storage provider for The Gaming Room's web-based multiplayer platform. AWS provides scalable infrastructure that integrates effectively with Linux server environments (Amazon Web Services [AWS], 2023a).

For static image assets, Amazon S3 provides durable, on-demand object storage designed for high availability and rapid retrieval (AWS, 2023b). Because Draw It or Lose It relies on rendering stock drawings during timed gameplay rounds, S3's distributed architecture supports consistent performance as demand increases.

For structured application data, including unique game names, team names, player identifiers, and session records, Amazon RDS is recommended. RDS supports relational database engines with mechanisms such as primary keys and unique indexes to enforce data integrity constraints (AWS, 2023c). Its pay-as-you-go pricing model supports cost control while allowing scalable expansion.

4. **Memory Management**:

When deployed in a cloud environment, Linux uses a virtual memory management system to efficiently allocate and protect system resources. Each process runs in its own protected memory space, and the kernel manages how memory is assigned and mapped to physical RAM. This isolation prevents concurrent game sessions and user requests from interfering with one another.

Linux uses demand paging, meaning memory is allocated only when required. For Draw It or Lose It, this allows gameplay logic and user sessions to consume memory dynamically rather than reserving unnecessary resources. Linux also maintains a page cache that stores frequently accessed data in memory, improving response time during timed gameplay rounds (The Linux Kernel Documentation, n.d.).

5. **Distributed Systems and Networks**:

To support communication across multiple platforms, Draw It or Lose It will use a distributed client–server architecture. Users access the application through web browsers, while core game logic executes on a centralized Linux-based server hosted in AWS. Communication occurs over HTTP and HTTPS using the TCP/IP protocol suite (The Linux Kernel Documentation, n.d.).

Within this architecture, the client manages user interaction while the server handles game logic, validation, timing, and data retrieval. Structured data is stored in Amazon RDS, and static image assets are retrieved from Amazon S3 through secure AWS network connections.

Because communication occurs over the internet, system performance depends on network reliability. Interruptions may temporarily affect gameplay. AWS mitigates these risks through redundancy, load balancing, and multi–availability zone deployments that support fault tolerance (AWS, 2023a). Managed services such as RDS and S3 further reduce risk through replication and automated backup capabilities (AWS, 2023b; AWS, 2023c).

6. **Security**:

Data transmitted between user devices and the Linux-based server will be encrypted using HTTPS with Transport Layer Security (TLS), protecting communication from interception or tampering. Within AWS, Virtual Private Clouds, security groups, and firewall rules restrict network access to authorized traffic only (AWS, 2023a).

On the server side, Linux enforces security through user and group permission models, file access controls, and process isolation mechanisms (The Linux Kernel Documentation, n.d.). These protections prevent unauthorized access to system resources and protect application processes from interference. Role-based access control further limits administrative privileges to authorized users.

For data at rest, Amazon RDS and Amazon S3 support encryption using AWS-managed keys, protecting structured game data and stored image assets (AWS, 2023b; AWS, 2023c). Automated backups and multi-zone replication improve resilience against data loss. Together, encrypted communication, Linux system protections, and AWS-managed security services ensure confidentiality, integrity, and availability across the distributed platform.

References

Amazon Web Services. (2023a). Overview of Amazon Web Services. https://aws.amazon.com

Amazon Web Services. (2023b). Amazon S3 features. https://aws.amazon.com/s3/

Amazon Web Services. (2023c). Amazon RDS features. https://aws.amazon.com/rds/

The Linux Foundation. (n.d.). What is Linux? https://www.linuxfoundation.org

The Linux Kernel Documentation. (n.d.). Linux kernel documentation. https://www.kernel.org/doc/html/latest/