The Gaming Room

CS 230 Project Software Design Template

Version 1.1

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	03/23/2025	Tristyn Tate	The cover page, document revision history, executive summary, design constraints, system architecture view, domain model, and recommendations were updated.
1.1	04/06/2025	Tristyn Tate	Reviewed previous content and updated evaluations and requirements.
1.2	04/19/2025	Tristyn Tate	Architecture recommendation.

Executive Summary

Creative Technology Solutions (CTS) is developing a web-based version of Draw It or Lose It for The Gaming Room. The game, similar to Win, Lose or Draw, involves teams guessing images from a stock library over four one-minute rounds. If the primary team fails to guess in time, opposing teams get 15 seconds to answer. The game needs to support multiple teams with unique names, only one game running at a time, and special IDs for tracking. Since The Gaming Room isn't sure how to set it up, CTS will handle the process and create a guide. Hardware decisions will come later, and a manager will check progress.

Requirements

The Gaming Room's web-based game must support multiple teams, each with multiple players. Team and game names must be unique, allowing users to check availability. Only one game instance can exist at a time, enforced through unique identifiers for games, teams, and players.

Design Constraints

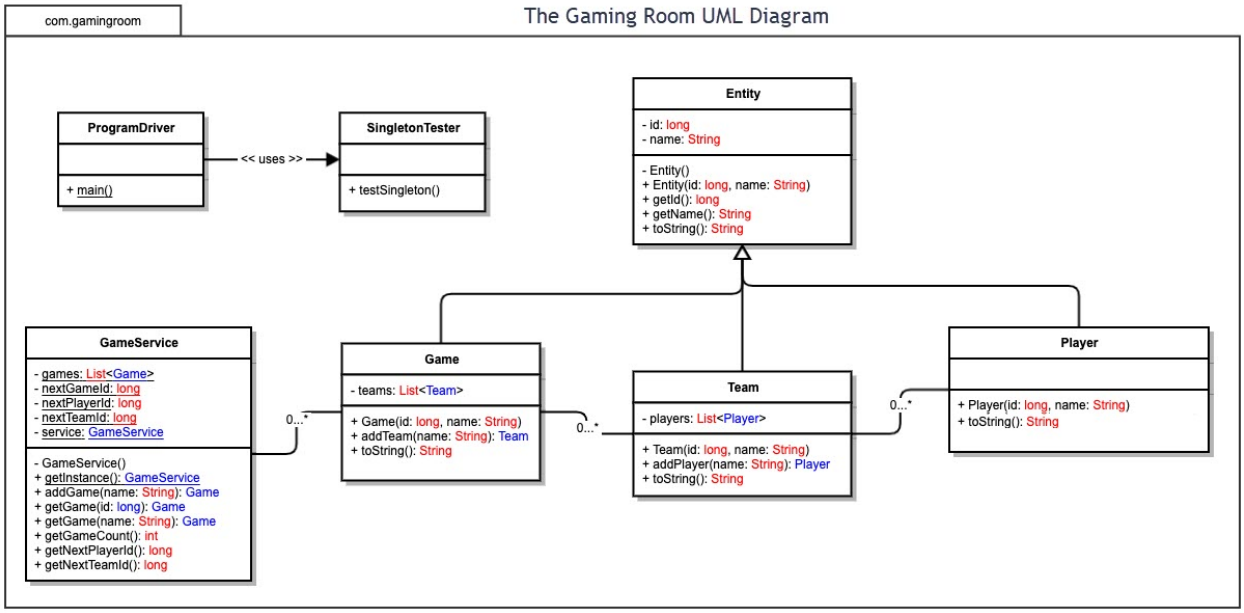
- The application must function across different platforms and devices.
- The system must be able to handle multiple games.
- Only one instance of the game can exist at any time.

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 UML diagram shows how Entity, Game, Team, and Player are connected. Entity is the main class that shares common traits with the others (inheritance), so there's less repetition. The game system follows a "has-a" setup (aggregation): GameService has Games, Games have Teams, and Teams have Players. This keeps data organized (encapsulation) and ensures the game runs smoothly with multiple teams, unique names, and only one active game at a time.



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 is suitable for the development and testing of web-based applications but is less commonly used for production hosting due to its higher cost, limited server resources, and scalability concerns. It supports server configurations through Apache or third-party services but is less prevalent in enterprise environments.	Linux is a great option for hosting websites. It's reliable, secure, and free to use. It works well with tools like Apache and Node.js, but can be tricky for beginners and needs regular upkeep.	Windows is a strong choice for web hosting, particularly in environments where Microsoft technologies are heavily used. It supports IIS, Apache, and Node.js, and is common in corporate settings with strong support for .NET. However, its higher cost and resource demands may make it less ideal for certain use cases compared to Linux.	Mobile devices offer portability and a large user base but present challenges in performance, screen size, and security when hosting web-based applications. They connect to the backend via RESTful APIs or WebSockets, but are not used for server hosting.
Client Side	Developing for Mac requires careful planning in terms of cost and time, with expertise in macOS development and cross-platform tools being essential to meet diverse client requirements. It is compatible with Safari, Chrome, and Firefox, and requires responsive web design for optimal user experience.	Fully compatible via web browsers (Chrome, Firefox, etc.) on Linux desktops. Developing for multiple clients on Linux is cost-effective but may require additional resources for infrastructure and maintenance. Developers must have expertise in Linux systems and tools to ensure compatibility, scalability, and security.	Windows is cost-effective and efficient but requires expertise in Windows tools and cross-platform compatibility. It supports Edge, Chrome, and Firefox, with a need for responsive design. development	Supporting mobile devices requires balancing cost, time, and expertise. Developers must optimize the web interface for mobile screens to ensure a seamless user experience across platforms.
Development Tools	Frontend development uses languages like HTML, CSS, and JavaScript, along	Frontend development utilizes HTML, CSS, and JavaScript, often supported	Frontend development uses languages like HTML, CSS, and JavaScript, along	Frontend development uses languages like HTML, CSS, and JavaScript, along

Recommendations

1. **Operating Platform**: I recommend using a cloud platform like Amazon Web Services (AWS) or Google Cloud. These platforms are powerful, flexible, and make it easy to run the game on phones, computers, and other devices. As more people play the game, the cloud can grow with it, so you won't have to worry about it slowing down or crashing.

2. **Operating Systems Architectures**: Cloud platforms work by spreading tasks across different parts — like web servers, databases, and traffic managers — kind of like a team where each person does their own job. This setup helps the game run smoothly on all types of devices, without getting overwhelmed.

3. **Storage Management**: To keep track of game data (like player info and scores), we can use cloud databases like Amazon RDS or Google Cloud SQL. These are reliable and can automatically back up data. For big files like drawings or images, we can use cloud storage (like AWS S3), which is like an online locker for large files.

4. **Memory Management**: Cloud platforms are smart — they automatically give the game more memory when lots of people are playing and scale it back down when it's quiet. This helps the game run fast and smooth without using too much power or resources.

5. **Distributed Systems and Networks**: To let the game work on lots of devices at the same time, we can use tools like APIs or WebSockets that let devices talk to each other in real time. If there's ever a problem, the cloud can handle it by rerouting traffic or using backup systems to keep the game online.

6. **Security**: Keeping player info safe is super important. We'll use encryption to protect data when it's being sent and stored. We'll also add things like password protection, multi-factor login, and limit who has access to what data. AWS and Google Cloud both follow high-level security rules to keep everything locked down and safe.