Draw It or Lose It

CS 230 Project Software Design Template

Version 1.2

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	01/22/2026	Shirl Lakeway	Initial completed draft of design template document
1.1	02/08/2026	Shirl Lakeway	Added development evaluation info
1.2	2/16/2026	Shirl Lakeway	Added final 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

The Gaming Room is looking to create a web-based version of their Android app, Draw It or Lose It. Prior communication has noted that The Gaming Room is lacking the expertise to setup the environment for this new game and is employing the services of Creative Technology Solutions (CTS) to streamline the development of this new application for the web. The new web-based game is planning to capture a larger audience for The Gaming Room which plans to be multi-platform.

Requirements

The Gaming Room has noted the following software requirements for the new web-based game:

- A web-based game that is supported on multiple platforms.
- A game will have the ability to have one or more teams involved.
- Each team will have multiple players assigned to it.
- Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
- Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.

Design Constraints

There are several design constraints with creating a web-based application. For The Gaming Room, this will require new infrastructure to host and support the client-server architecture for the application. This means that investments into one or more cloud providers or hosting on premise hardware will be required. This infrastructure is needed to manage the game connections, game states and storing player information to handle authentication. This will help to ensure that games and team names are unique, games will have the ability to have one more team involved and teams having multiple players assigned to them as noted as one of the software requirements.

Next, considering that we will be developing this new game as a web-based application, the game must be able to run on different web browsers and platforms. This will require a responsive UI design to adapt to different screen sizes and varying inputs. To coincide with this, there can only be a single game instance in memory which will need a database or an authentication method to ensure correct game states. Implementing a singleton patten will help to limit multiple game instances running in memory.

Lastly, we will want to develop this new web-based game to be extensible and maintainable for future updates to the game. This will encourage designing the code base to be as modular as possible to make adding and changing aspects of the game easy to integrate.

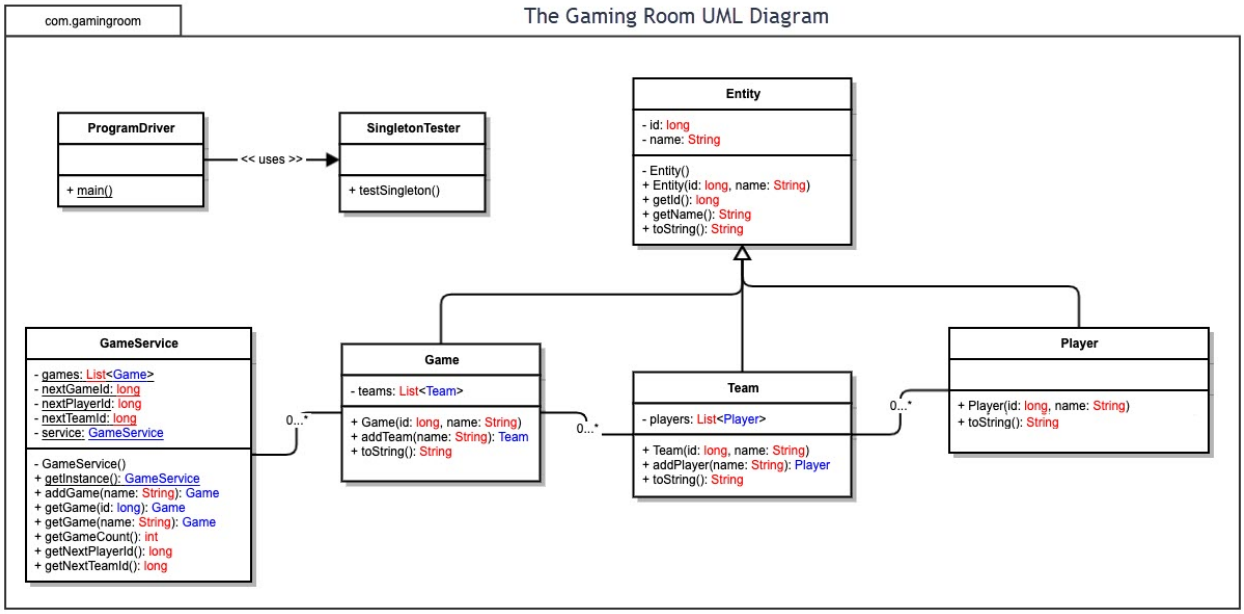
System Architecture View

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 included below for The Gaming Room utilizes a total of 7 classes. The Entity class serves as the parent class for the Game, Team and Player which are the child classes. Within the Entity class, shared attributes of id and name are defined which also provides methods used by each of the other child classes. Game, Team and Player classes inherit from Entity which then each of the child classes have additional methods to extend functionality. Game class will contain information about the teams created by users, Team class will contain information about the players. GameService manages all the game sessions and uses the singleton pattern to verify that only one instance of a game exists in memory which creating unique IDs.

Private fields are used to control access and to encapsulate values which will be access though public methods. This will help to maintain the integrity of the players and game instances. Using the Entity class, this will abstract common values shared between all the game functions and objects which will help making managing the game for future updates easier. Using the Game class which stores information of teams and then the Team class which contains player information, using this composition principal of will help to represent real-world relationships. Lastly, the singleton pattern in GameService will ensure that only of instance managing all the games exists in memory.



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	Characteristics of MacOS is that it is unix-based which is typically a stable and secure operating system but is rarely used as a production server environment. Advantages of using MacOS as the server-side environment is that there is good developer tooling and support, POSIX compliance and good development mirroring to Linux platforms. Disadvantages of using MacOS for the server side is the lack of server-grade hardware options, higher costs and low industry adoption for hosting this specific platform.	Characteristics of Linux is that it is the most widely adopted operating system for a production server environment. It is open-source, modular and customizable meaning high degree of flexibility in hosting and managing a web-based application. Advantages of using Linux as the server-side environment is performance, security, open-source support, low cost and compatibility with all server stacks. Disadvantages of using Linux for the server side is that it does require a more specialized knowledge to properly administer, configure and maintain.	Characteristics of Windows is that it is a closed-sourced operating system, but it has strong enterprise support, a dedicated Windows Server operating system and most software made to run on Windows Server has a GUI for management. Advantages of using Windows as the server-side environment is that it is made for enterprise environment meaning reliability and professional support is available. There is an incredible number of resources available to support a web-based applications running on Windows Servers. Disadvantages of using Windows Servers is a high and confusing licensing model, less efficiency and fewer open-source tools to support deployments.	Characteristics of mobile devices is that they are typically not used to host server workloads. While they do offer a portable device, they are typically used for testing server deployments. Advantages of using mobile devices would be that they are affordable to buy in bulk and offer a way to host a server without inventing in wiring infrastructure. Disadvantages of using mobile phones is that is it not industry standard, lack of robust processing power, battery considerations and not enterprise-grade operating system for management, configuration and deployments for web-based applications.

Client Side	Development and testing for MacOS clients requires having Mac computers which presents a high cost for equipment. Virtualization options for MacOS are limited as well. There is also a cost for the Apple App Store for publication of the application. Developers will need experience with Swift/Objective-C, Xcode, and macOS UI/UX conventions as these are platform specific tools. MacOS does have a smaller market share of devices compared to Windows so that means there are potentially less customers on MacOS.	Development and testing for Linux is widely accepted and well documented. There is a low cost for entry to start development with Linux and there is a plethora of close and open-source tools to aid in development for client side. Linux does encompass many different distributions or "flavors" of Linux which can complicate testing to ensure applications work on all distribution and package managers. Developers will need strong command-line and system-level knowledge along with familiarity with Linux filesystem, permissions, and package ecosystems. Linux remains to be the smallest market share of devices. Unless there is a particular need or desire to support a Linux support, is not recommended.	Development and testing for Windows is well documented and extensible. While there is a moderate cost to purchase devices, virtualization of Windows for testing is highly recommended. There is a wide range of different tools available to aid in in development ranging in costs. Windows OS has the largest market share of devices so developers will have ample resources and generally more familiarity developing for Windows. This means supporting development for Windows will be the easiest to achieve. Windows accounts for about 70% of all computer users. Prioritizing development for Windows will ensure the largest audience available for the application.	Development and testing for mobile devices is well documented and relatively straightforward. It requires purchasing developer licenses for app stores and obtaining multiple different devices for testing. Developing for both iOS and Android will typically double developing timelines. There are frameworks that can help reduce this to help speed up releases. Native development requires Swift/Objective-C for iOS and Kotlin/Java for Android which is well documented. Cross-platform development requires expertise in frameworks like Flutter, React Native, or Unity. iOS and Android account for hundreds of millions of devices and users so the time and effort to develop for these platforms is well worth any costs.

Development Tools	Macs use Objective C and Swift as the development languages of choice though C/C++, Javascript, Typescript and Python are supported. Xcode is the primary IDE for developing for Macs which is a $99 USD cost per developer per year. Xcode does offer tools specific for testing applications that would run on Mac OS.	Linux developing is very flexible in that it can use C/C++, Java, Python or Rust to name predominant languages. Many IDEs are supported on Linux like Visual Studio and Intellij or open-source options like Vim and Emacs text editors. This offers developments many flexible options for how they would like to develop applications.	Windows development is predominantly done with C# and .NET languages. Other languages like Java and C/C++ are available but C# and .NET are the best options for Windows development. Visual Studio is regarded as the most preferred IDE when developing on Windows as this is a Microsoft software which integrates well with Windows and the Microsoft ecosystem. Visual Studio ranges from $45 to $500 a month per developer but Visual Studio Code is a free option that can meet most developers' requirements for a Windows IDE.	Mobile app development will use Objective C and Swift for iOS and Java for Android devices. Whereas with the other platforms, developing for mobile devices will require expertise in multiple different languages. Android devices use Android Studio, which is free to download, and iOS will use Xcode which is $99 per developer per year. Additionally, there are yearly costs associated with publicizing apps to each app store.

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**: The optimal operating platform that The Gaming Room should build around should be a Linux-based environment. Selecting either Ubuntu Server or, for more options for enterprise support, Red Hat Enterprise Linux would be the strongest options for the environment. Both options offer scalable solutions, plethora of web and application server stacks, mature security controls and broad support from cloud platforms to host and manage the environment. This choice of deploying the server and application stack on Linux-based operating systems allows The Gaming Room to serve modern, responsive web clients on any desktop OS and mobile OS though standard-based HTTPS APIs and browser-based interfaces. Additionally, this offers operational cost saving methods in terms of not purchasing licensing for servers. There are support licenses offered from Canonical and Red Hat to help support the operating systems but that is an optional cost.

2. **Operating Systems Architectures**: As mentioned, using a Linux-based platform is the recommended architecture that The Gaming Room should invest into. Linux offers complete control over the user space and kernel space on the operation platform. This control allows administrators to fine tune performance and low-level concerns like process schedule, memory management, drivers and networking controls. The control offered on Linux far exceeds what is offered by other operating system platforms from Microsoft for example. In the user-space for Linux, services can host the web server, application runtimes and database connections. This approach supports a layered web-application design with front-end services, stateless application services and other data management services. Lastly, containerization with Docker or Kubernetes is recommended as this allow for developers to quickly create new instances of the Draw It or Lose It server stack to help test and deploy new features or scales out when server demand grows for the game.

3. **Storage Management**: Linux offers several relational based management systems for a database like PostgeSQL and MySQL which can be ran as a server or as a cloud managed database service for The Gaming Room. Using a virtual file system with journaling will offer a reliable and expandable server platform for storage. Ensuing that the data stored on the server for the Draw It or Lose It application is accurate, verifiable and auditable is important as this is how user state and progress is maintained. A separate server should be built for the images and other resources that are not to be stored within the application database. This separation of functions allows the management plane to be simple and makes backup, snapshots and replication simpler. Breaking out the different types of storage functions to their own servers additionally makes scaling the infrastructure easier as well.

4. **Memory Management**: Linux utilizes many different methods for managing memory on the operating system. Specifically, Linux uses virtual memory, paging and on-demand loading to manage RAM which the Draw It or Lose It application can take advantage of. For each of the components of the server stack, web server, application runtimes and database, it can run each process in its own address space which protects it from memory leaks and unintended manipulations. The Linux kernel uses paging and caching which assists frequently access code and data in memory and removing those that are not. Draw It or Lose It will have instances for

each game which will use transient memory for game states and for persistent states for users like status and records, they would be recorded into the database. This memory management method will allow for the operating system and processes to scale its memory usage as more customers play Draw It or Lose It.

5. **Distributed Systems and Networks**: Draw It or Lose It should be architected as a distributed, service-orientated web application. The main logic and APIs for the application will run on Linux servers which will be exposed publicly over HTTPS as RESTful endpoints. Clients will then communicate to these endpoints to create and join games, submit answers and receive game state updates in real time. In front of these endpoints will be load balancers to have manage traffic across different instances and help to maintain a good user experience. These load balancers will have health checks and auto-scaling functions to help address demand and periods of low activity. Alongside the servers for game logic and APIs will be database servers and caches to update game and player states as well as store image files and other data for Draw It or Lose It. This will help to make the application resilient, upgradable, and scalable as outages and new features are released while minimizing downtime and outages.

6. **Security**: Linux offers many features that provide a robust user and group permissions, network controls though firewalling and regular security updates for the operating system. It is recommended that all client-server communication use HTTPS and TLS to protect data in transit between the endpoints. User information such as password must be stored as hashed values with encryption and access controls to the databases must be tightly controlled and monitored. Authentication and authorization should utilize either token-based or session management such as JSON web tokens with short lifetimes or refresh tokens. This helps to confirm user's identity and prevents malicious actors from compromising the platform. Lastly, input validation, parameterized queries and secure coding practices should be enforced to protect against common web vulnerabilities. All this combined should have clients require the most minimal amount of user and game data on their device. When data is stored on a client endpoint, it should be obfuscated to make reverse engineering the application as difficult as possible. By releasing routine updates for Draw It or Lose It to enhance security, this will position The Gaming Room well to protect itself and its players from threat actors.