

“Draw It or Lose It”

# **CS 230 Project Software Design Template**

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 8**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 05/20/2023 | Benjamin Dowell | Updated Executive Summary, Requirements, Design Constraints and Domain Model |
| 1.1 | 06/04/2023 | Benjamin Dowell | Updated Evaluation section |

**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](#_sbfa50wo7nsh)

The Gaming Room aims to create a cross-platform web game titled "Draw It or Lose It," which is currently exclusive to Android. This game involves multiple teams, each comprising several individuals, engaging in four rounds lasting one minute each. In each round, a picture is selected from a library of images, and one team attempts to guess it within the given time limit. If the picture remains unsolved, members from the opposing teams take turns answering within a 15-second time frame.

## 

## Requirements

To ensure successful code and software development, it is important to consider the following requirements. While focusing on the game aspect, we also need to address the broader application development aspects. The Gaming Room aims to make the game accessible on all devices, which includes expanding its availability beyond the Android platform. This entails integrating the game into other mobile devices, such as those running Apple's operating system, as well as desktop machines running Windows and Linux.

To achieve this compatibility, we have two options. Firstly, we can rewrite the existing code for each version we want to run on. This would allow the game to run natively on other-than-Andriod devices. Alternatively, we can explore the possibility of leveraging existing code and incorporating other languages to ensure compatibility across different platforms. By utilizing a combination of programming languages, we can create a robust and adaptable codebase that can be executed on various devices.

By fulfilling these requirements, we can develop a versatile game application that can be enjoyed by users on multiple platforms, catering to a wider audience and enhancing their gaming experience.

## [Design Constraints](#_2et92p0)

Run seamlessly on multiple platforms, ensuring broad accessibility for users.

Ensure that each team consists of multiple people.

Support the participation of one or more teams.

Implement a system where both game and team names are unique, enabling users to verify name availability.

Restrict the existence of only a single instance of the game at any given time.

## [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 Entity class establishes a relationship between the Game, Team, and Player classes, enabling them to inherit or obtain information from the Entity class. This relationship is depicted through inheritance in the UML diagram, illustrating that each class shares common attributes such as "name" and "id". Consequently, the Entity class acts as a superclass.

Examining their relationships, we observe that Team and Player have a "has a" relationship type, indicating that they possess instances of other classes. On the other hand, Game has a Team, and GameService has Games, which are examples of aggregation (HAS-A) relationships. In this context, when a user "has a," it signifies that an instance of one class contains a reference to an instance of another class.

Analyzing the UML diagram, I see that GameService holds a reference to Games, Games holds a reference to Team, and Team holds a reference to Player. These relationships illustrate how the classes are interconnected, allowing for the efficient organization and retrieval of data within the game 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)

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: Unified development environment, comprising of multiple OS within Apple.  Advantages: Familiarity and ease of use. For those experienced with Apple, one is like the other and each are seamless together.  Weaknesses: Niche development options, though vast, limited to the Apple environment. Cost is a limited factor as well since Apple tends to be on the higher. | Characteristics: Linux is an open-source operating system. Linux is also known to be very stable and with optimal performance.  Advantages: Being open source, Linux can be a very cost-effective avenue. There is large amount of software and tools for this same reason.  Weaknesses: Some proprietary software or hardware may not be fully supported. | Characteristics:  Very widely used user-friendly interface. Because of its wide acceptance, Windows is broadly compatible with many software and hardware.  Advantages: Familiarity due to dominant market share. Windows has a vast amount of development tools.  Weaknesses: Being popular creates security concerns as it is the most targeted for malware and security threats. Could be limited in customization versus open-source options. | Characteristics: As implied, portability is the leading characteristic as well as being generally touch enabled these days.  Advantages: Mobile devices are widespread and enhance user engagement. Most people own a mobile device while some own multiple. Mobile devices engage the user by location services, camera access, and other personalized experiences for web applications.  Weaknesses: Limited on screen size as well as mobile device size limiting performance. |
| **Client Side** | Cost: Development tools and hardware. As with all things, if needing specific software developing needs, Licenses and tools can become expensive. Hardware is also a bit pricey.  Time: If you are not familiar with Mac and Apple in general, there may be a learning curve.  Expertise: Being quite isolated from other development environments, expertise within Mac are needed when serving multiple clients. The knowledge needed for cross-platform development even within Apple is specific to the brand to ensure consistent functionality. | Cost: Being that Linux is free and open-source, cost is quite low  Time: May have to spend time learning platform-specific areas. Given the open source, you may spend more time testing compatibility as well.  Expertise:  A unique OS requires unique knowledge in developing on the Linux platform, compatibility in cross-platform development and general configurations, security and troubleshooting on the LinuxOS. | Cost: Licenses can be a huge upfront cost. As with Apple, development tools are vast but many cost money.  Time: Higher chance of qualified individuals having Windows knowledge due to popularity. Compatibility testing may be needed between Windows variations.  Expertise: Developers will need Windows knowledge for developing. | Cost: Software Development Kits (SDKs) may require upfront costs  Time: Given the wide number of different mobile devices, a lot of time can be spent on compatibility testing for screen resolutions and hardware configurations. Platform specific development such as Apple and Android.  Expertise: Working on mobile devices generally require platform-specific development skills given various developers and hardwares. |
| **Development Tools** | Programming languages include Swift and Objective-C. Swift is the primary programming language used for Apple, while Objective-C is still used and is a superset of C geared towards Apple.  Integrated Development Environments (IDEs): Apple’s official IDE for MacOS and iOS is Xcode.  Other notable tools include Interface Builder and Homebrew. Interface Builder is a visual editor integrated into Xcode and Homebrew is a manager that allows developers to install and manage various open-source software and libraries. | Programming languages include Python, JavaScript and Ruby. Python is a widely used programming language that is popular on Linux as well. JavaScript is used for web development and generally used on the client-side. Ruby is also used for web development.  IDEs: Some Linux IDEs include Visual Studio Code and JetBrains IDEs. VSC is a popular cross-platform code editor and JetBrains provides a slew of IDEs such as: PyCharm, WebStorm and RubyMine.  Other tools include Git for managing source code and MySQL for open-source relational database management systems. | Programming languages include C# and JavaScript. C# is widely used by Microsoft for programming and JavaScript is used for web development in Windows.  IDEs: Microsofts own IDE is Visual Studio. It is a comprehensive product with many developer’s tools.  Other tools include .NET Framework and Internet Information Services (IIS). .NET Framework are development frameworks for building Windows applications. IIS is a web server provider commonly used for hosting web applications. | Programming languages include Swift, Objective-C, Java, and Kotlin. Swift and Objective-C are used in Apple development as discussed under Mac. Java and Kotlin are Android’s traditional and modern programming language respectively.  IDEs: For Apple, Xcode is the way to go and for Android, Android Studio is the official IDE for development.  Other tools include React Native and Flutter. React Native is a cross-platform framework that allows developers to build using JavaScript and React. Flutter is also a cross-platform framework developed by Google. |

## 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**: I recommend the operating platform Unity. Unity is already built for cross-platform implementation. You can write code once minimizing the need for specific catering to each platform. Unity supports a wide range of hardware, including smartphones, tablets, PCs, consoles, AR and VR. Given these options, The Gaming Room can further expand seamlessly in the future.
2. **Operating Systems Architectures**: Unity provides an asset store with a vast library of pre-built scripts, models, textures, and animations. This can significantly speed up development, lending more time to cross-platform testing and adaptation. Unity supports Windows running on x86 and x64 (Windows 7, Windows 8, and Windows 10), x86 and x64 MacOS and Linux (Mojave, Catalina, Big Sur, Ubuntu, Fedora and CentOS), Android (ARMv7 and ARM64), iOS ARM64 (iPhones and iPads), Consoles (PlayStation, Xbox, and Switch), and web browsers with WebGL technology.
3. **Storage Management**: Pairing Unity’s built-in features along with an external database system is key. Unity provides built-in features that can handle storage management for various purposes, such as asset management and resource management. Unity’s asset management can import, organize, and manage game assets such as textures, models, and more. Unity can help optimize storage by loading only the necessary assets at runtime. Unity’s resource management can dynamically load and unload assets during gameplay, which is highly desirable for Draw It or Lose It. For external database solutions, Unity supports various database solutions, with SQLite being the most popular. The complex storage for managing user profiles, game progress, and leaderboards can be managed externally within Unity to store and query structured data efficiently.
4. **Memory Management**: Unity’s memory management techniques are perfect for Draw It or Lose It due to the high-resolution images needed for the game. Unity employs automatic memory management through a garbage collection system. This automatically detects and frees memory that is no longer in use and will help the game run more efficiently. Unity also provides mechanisms to unload unused resources during gameplay. In the instance of deploying high-resolution pictures, it is best to keep the memory free of clutter.
5. **Distributed Systems and Networks**: Establishing network connectivity is crucial for enabling communication between platforms. The server hosing the game should have access to the internet. Distributed systems involve breaking down the game’s functionality into components that can run on different platforms and communicate with each other. Game clients are applications running on various platforms where players interact with the game, each game client should render the game, handle user input and communicate with the game server. The game server acts as a central hub coordinating the game, facilitating communication between clients, manages game logic, enforces rules, and handles synchronization between game clients. When designing the distributed software and network architecture, it is essential to consider dependencies between components and plan for potential issues. Connectivity can be affected by various factors, including internet outages, network congestion, or device-specific issues. The system should allow players to reconnect easily and reinstate synchronization.
6. **Security**: Unity supports encryption protocols and secure communications to protect user information between game and server clients. Unity can also implement user authentication with the game to verify the identity of players through methods such as OAuth, JWT, or custom token-based authentication. Collecting and storing only necessary user information, known as data minimization, is always a helpful principle when thinking about security. In respect to data storage, Unity’s PlayerPrefs tool uses encryption for storing small amounts of sensitive data as well. With a widely used platform, you get ongoing security monitoring and updates to stay proactive in monitoring vulnerabilities and updates relevant to the platforms and services you use.