

Draw It or Lose It

# **CS 230 Project Software Design**

Version 1.2

## 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 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 07/19/2024 | Kristen Hernandez | Addition: Initial design project information  Addition: Information for evaluation section of project  Addition: recommendations |
| 1.1 | 07/31/2024 | Kristen Hernandez | Evaluation of various platforms’ hosting characteristics |
| 1.2 | 08/12/2024 | Kristen Hernandez | Recommendations updated. |

**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)

Our new client, The Gaming Room, aims to develop a web-based game accessible across multiple platforms. They want to base this game on their existing Android game, Draw It or Lose It, which involves teams competing to guess images being drawn within timed rounds. Currently, the game only runs on Android, but the goal is to make it available on other operating systems such as MacOS, Linux, and Windows. The game comprises four rounds, each lasting one minute.

## Requirements

* Game must be accessible via web-based platforms in order to ensure compatibility among different operating systems and devices
* Each game should accommodate one or more teams, allowing for multiple players to be assigned to each team
* Game and team names should be unique to avoid conflicts, and users should be able to check name availability during each team creation
* The application must enforce a single instance of the game in memory at any given time, achieved by implementing unique identifiers for games, teams, and players
* If a team fails to guess the puzzle within the allotted time, the remaining teams should be given one guess each within a 15-second time limit to solve the puzzle
* Game rounds should have specific time limits, such as one minute per round, with drawings gradually revealed until fully visible at the 30-second mark

## [Design Constraints](#_2et92p0)

* Cross-Platform Compatibility: Since the game is currently an Android app, we’ll need to convert it into a web app to ensure it can be accessed from various operating systems. Using a REST API to communicate via HTTP will help to avoid some issues with compatibility.
* User Interface: While we could replicate the existing Android app interface, we also have the option to enhance the design and release it as a new game.
* Support for Multiple Teams and Players: The game must accommodate multiple teams, each with several players. This requires a client-server architecture capable of handling many concurrent players.
* Unique Identifiers: Each game, team, and player must have unique IDs to ensure that only one game instance exists in memory at any time. We need to manage memory allocation efficiently and identify players’ platform-specific IDs.
* Images and Copyrights: The Gaming Room wants to use images in the Draw It or Lose It game. We will need to adopt the current Android platform images and ensure their compatibility with other platforms. Additionally, we must secure licenses or copyright permissions for any new images.

## [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)

We will use a Unified Modeling Language (UML) diagram to visually design the game system. This diagram will illustrate the relationships between the game, team, and player classes with each class inheriting attributes from a superclass. The ProgramDriver class, shown in the top left corner, will use the SingletonTester to ensure only one game instance exists in memory. The GameService class will contain the complex methods forming the game’s backbone and functionality. Unique IDs for games, teams, and players are required, which is why dedicated classes are designed in the diagram. The connections between classes represent associations, with numbers indicating the quantity of associations. For example, the GameService class can be associated with zero or more Game classes. This diagram will guide the development process, with adjustments made as necessary to achieve the final product.

**"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** | Apple, the company that developed MacOS, provides server-based environments. Utilizing a MacOS server offers significant benefits for Mac clients on a network. These advantages include full support for Mac applications and straightforward general administration through an excellent graphical interface. However, there are some drawbacks to using a Mac server. Firstly, maintenance can become costly. Secondly, it is not ideal for large companies or enterprises that depend heavily on third-party programs and customizations | Linux offers advantages for web hosting. Major companies like Google even use Linux servers. The most notable benefit of a Linux server is that it is free and open-source, making the deployment and use of tools inexpensive or free. Additionally, Linux is highly secure and customizable, allowing organizations to tailor security measures to their specific needs. Linux web hosting supports various programming languages, including Python, PHP, Perl, and Ruby. However, there are some disadvantages to using a Linux server. The steep learning curve for those without prior experience. Also, some applications might not be supported, and migrating from Windows to Linux can present challenges. | Windows is a versatile operating system, but it is proprietary software, meaning licensing fees are required. Although Windows licensing can be costly, there are several benefits to using a Windows web host server. One advantage is its extensive support for a wide range of applications and third-party software. Another benefit is the ease of applying patch updates and hardware upgrades. Additionally, for developers familiar with the platform, scripting languages like ASP.NET and databases such as MySQL are fully supported. | While using mobile devices for web servers is not well-known and may seem impractical, it is feasible to implement. Oracle is one of the companies that provide mobile server-side solutions. Oracle Database Mobile Server can manage applications, users, devices, and data across large deployments of mobile or remote devices. The advantages of Oracle's database structure include support for iOS and Android development tools, as well as synchronization with Oracle NoSQL. |
| **Client Side** | Costs include acquiring development tools, Mac hardware, and developer salaries, along with ongoing maintenance expenses. Development time is extended due to the need for extensive testing, a potential learning curve for inexperienced teams, and the necessity of regular updates to stay compatible with Apple's frequent OS changes. Expertise required includes proficiency in Swift and Objective-C, UI/UX design according to Apple's guidelines, system integration, security best practices, and cross-platform development strategies. Proper planning and resource allocation are essential to address these considerations effectively. | Costs are generally lower due to Linux being open-source, but expenses may arise from specialized tools, hardware, and skilled developer salaries. Development time can be extensive due to the need for thorough testing across various distributions and configurations, as well as potential learning curves for developers unfamiliar with Linux. Expertise needed includes proficiency in languages like Python, PHP, and shell scripting, along with knowledge of Linux system administration, security best practices, and the ability to customize the OS for specific requirements. Effective resource allocation and planning are crucial to manage these aspects efficiently. | Costs include purchasing Windows licenses, development tools, and potentially higher developer salaries due to specialized skill requirements. Development time can be prolonged due to extensive testing across various Windows versions and ensuring compatibility with numerous third-party applications. Expertise required encompasses proficiency in languages like C#, familiarity with the .NET framework, and knowledge of Windows system administration, security protocols, and the intricacies of the Windows ecosystem. Efficient planning and resource allocation are essential to address these considerations effectively. | Costs include acquiring various mobile devices for testing, purchasing development tools, and potentially higher salaries for specialized mobile developers. Development time can be extended due to the need for thorough testing across different operating systems (iOS, Android), screen sizes, and device capabilities. Expertise required includes proficiency in languages like Swift and Kotlin, familiarity with mobile development frameworks, and knowledge of mobile UI/UX design principles, as well as app store submission processes and mobile security best practices. Effective resource allocation and planning are crucial to manage these aspects efficiently. |
| **Development Tools** | For deploying software on Mac, relevant programming languages include Swift and Objective-C. Essential tools and integrated development environments (IDEs) include Xcode, which is the primary IDE for MacOS and iOS development, offering a comprehensive suite for coding, testing, and debugging. Additional tools such as CocoaPods for dependency management, Homebrew for package management, and Git for version control are also commonly used. These languages and tools enable developers to create robust and efficient Mac applications while ensuring compatibility with Apple's ecosystem. | For deploying software on Linux, relevant programming languages include Python, C, C++, Java, and Shell scripting. Commonly used IDEs and tools are Eclipse, PyCharm, Visual Studio Code, and GNOME Builder. Additionally, tools like Git for version control, Make and CMake for build automation, and package managers like APT and Yum are essential for efficient development and deployment on Linux systems. | For deploying software on Windows, relevant programming languages include C#, C++, and Visual Basic. Key tools and IDEs used are Microsoft Visual Studio, which provides a comprehensive environment for development, testing, and debugging. Other useful tools include .NET framework for application development, PowerShell for scripting, and Git for version control. These languages and tools enable efficient creation and deployment of software tailored to the Windows operating system. | For deploying software on mobile devices, relevant programming languages include Swift and Objective-C for iOS, and Kotlin and Java for Android. Key tools and IDEs are Xcode for iOS development and Android Studio for Android development. Additional tools such as Flutter and React Native are used for cross-platform mobile development. Git for version control and various testing tools like XCTest and Espresso are also commonly used to ensure efficient and robust mobile application development. |

## 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**: Based on our evaluations, we recommend the Linux operating system. Linux is open-source, which reduces costs, supports various development tools, and is both flexible and secure.
2. **Operating Systems Architectures**: Most operating systems share similar architectures, acting as an interface between the user and the computer resources. These architectures include hardware, kernel, and shell. The hardware consists of memory, CPU, I/O, and other physical devices. The kernel acts as a bridge between applications and data processing at the hardware level. The shell provides an interface for users to interact with the operating system.
3. **Storage Management**: For the web-based game "Draw It or Lose It," we recommend using cloud storage. Cloud storage allows us to utilize only the necessary space and easily scale up as the game's popularity grows, avoiding the costs of additional hardware. Based on our evaluation, Google Cloud is the best fit for this project.
4. **Memory Management**: Linux is versatile and easy to customize. We will use Java for backend development, as it handles memory management automatically through a garbage collector, eliminating the need to implement memory management logic in our application.
5. **Distributed Systems and Networks**: Since the game is web-based and the Google Chrome browser is available on all platforms, users can play regardless of their operating system. This ensures global accessibility without OS-specific considerations.
6. **Security**: By utilizing Google Cloud services, we eliminate concerns about data center security, as Google provides the necessary hardware and infrastructure. For software security, we will use Google apps, ensure the game has automated patches and updates, and conduct QA testing before deployment.