

<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 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/13/23 | Michael Isenhour | Version 1.0 |

**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’s goal is to transform their Android game, “Draw It or Lose It,” into a web-based multi-platform experience. Currently it is only available on Android. In this game multiple teams compete trying to guess drawings that are pulled from an image library in four one-minute rounds. If the question goes unanswered, then the opposing team gets a 15-second chance to guess and score a point.

## [Design Constraints](#_2et92p0)

**Cross-Platform Compatibility**: The game currently runs on Android but must be compatible on other platforms. To achieve this, we should focus on responsive design and use HTML5 or JavaScript frameworks.

**Unique Naming System**: Each game and team must have a unique ID to avoid conflict. To achieve this, we can implement a naming convention that will generate unique identifiers and store them in a database in real-time.

**Images and Content Moderation**: The images used for this game must be stock images and royalty-free. To achieve this, we need to implement content moderation tools that would regularly review and update the image library to comply with copyright laws as well as ensure the content is appropriate.

**Single Game Instance**: Only one instance of the game should exist at any one time. To achieve this, we will rely on our Unique Naming System mentioned above as well as utilize the Singleton design pattern programming principle.

## [Domain Model](#_8h2ehzxfam4o)

From looking at the UML we can see that Entity creates a relationship between the Game, Team and Player classes, thus making Entity a superclass. We also see by the use of the arrow that each of these classes mentioned inherit from an attribute in Entity.

Another thing we notice from the UML is that the ProgramDriver class is the one running the program and also will use SingletonTester to test the programs code. The purpose of the SingletonTester is to restrict the game to allow only one instance of it to run so that we can test the code properly.

Lastly the GameService class will hold the meat of the program as it will be what makes up the programs functionality. The lines between GameService to Game shows association same as the lines between Game and Team, and Team and Player. They are all zero to many as a GameService can have zero or more games, a game can have zero or more teams, and a team can have zero or more players.

**"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** | Mac offers many useful tools when it comes to server-side development. It is also a very stable platform and can provide a reliable server environment. However, cost is this platforms downfall as server hardware and other services is costlier than the alternatives. | Linux server-side development offers cross-platform compatibility it is also very good at handling scalability which would be excellent for accommodating growing player numbers. However, those who aren’t skilled with Linux will find it extremely complex to set up and maintain the servers. | Like Linux Windows offers server-side cross-platform compatibility. One of its major advantages is that many developers are already familiar and skilled when it comes to Windows server environments which saves on having to train developers. Weaknesses however are Windows servers have licensing costs and while it is compatible with desktop environments it has limited reach on the mobile market platform. | Advantages to making it on Mobile Devices would be centralized logic meaning there would be consistent gameplay experience for all mobile users. However a weakness for developing server-side apps for mobile devices is having to jump through app store approval. |
| **Client Side** | Mac has a beautiful environment when it comes to game development. However, Mac has a smaller user base then Windows which will limit your games audience. | Linux is open-source allowing developers more flexibility and customization when it comes to client-side development. It also has a rich developer community for support. However due to it having many different Linux versions a developer can run into potential compatibility issues. | Windows biggest advantage is how many people are windows users. It provides a massive market reach and player base and also had robust tools and libraries for developers to use when it comes to client-side development. However it can be costly due to licensing costs and fees. | A major advantage to Mobile Devices is they are everywhere which leads to a huge player audience. They also have established app stores making it easy for game distribution and monetization. A major disadvantage however is with newer devices coming out yearly with varying specifications it could lead to device fragmentation and optimization challenges. |
| **Development Tools** | Tools that would be used for developing on the Mac would be Swift and Xcode. | For developing on Linux you can use languages such as Python and Java and IDE environments such as VS Code, eclipse or PyCharm. Due to Linux being open-source there are many tools and options available. | Like Linux there are many options available that you can use. When it comes to programming languages you can use Java, Python, or C/C++. As for tools and IDEs you can use Visual Studio and VS Code or Eclipse to name a few. | For Mobile Devices it would depend on if you want to have a split team where some develop for iOS (Swift) while others develop for Android (Kotlin) or if you want to use tools like React Native, Flutter, or Ionic to develop for both platforms at once. |

## 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**: When it comes to the operating platform that would work best, I would have to recommend Windows. Putting it on the Windows platform would instantly make it accessible to a vast audience and its already large gaming community. It would also make it developer friendly as there are many developer tools that are well supported on Windows.
2. **Operating Systems Architectures**: The systems architecture is broken down to older 32-bit computers and 64-bit newer modern computers. For maximum user compatibility and an overall broader market reach, I would support both architectures. This is because while in some regions newer hardware might still not be easily accessible you may still find users who would enjoy the game.
3. **Storage Management**: Windows uses NTFS as their file system for storing user data. The NTFS system supports large files and can handle large volumes of data which would be great for the games data storage. Also, it has security features such as file encryption. However, you can also use a database system such as Microsoft SQL for more structured data storage. Microsoft SQL is perfect for its scalability and security features it offers. It also would be able to handle queries and game related data like the players profiles, game stats, and images.
4. **Memory Management**: When it comes to Windows memory management it handles memory allocation extremely well. The Windows OS has advanced virtual memory techniques which allows disk space to be used as virtual RAM. This feature increases the memory for applications such as games to help them run smoother. Windows also is capable of memory allocation and deallocation based on the applications needs. However, developers can also focus on code optimization practices to help with memory efficiency. These practices could be making sure to avoid memory leaks and using garbage collection efficiently.
5. **Distributed Systems and Networks**: For cross-platform requests you could develop RESTful APIs. Using RESTful APIs you could integrate features for this game such as multiplayer functionality and matchmaking as well as leaderboards. You could also implement WebSocket for real-time communication between clients and live game updates. For scalable cloud infrastructure and smoother gameplay, you can use AWS or Azure. These cloud services offer high performance computing and database management which would greatly enhance the games overall performance and reliability.
6. **Security**: Windows has built in security features and implements encryption, data validation and software updates to protect its userbase. For its security features Windows uses Windows Defender and Firewalls to help protect its system from hacking and other attacks such as malware. These security features being in place would allow your users to enjoy their gaming experience securely. Also mentioned above, Windows NTFS uses encryption for secure data transmission which would be effective to guard sensitive player information such as their game data, profiles, etc. while both at rest and in transit. Windows also offers patch updates to their system when needed to keep their systems secure and prevent breaches and has many system security monitoring tools that users can download and monitor their systems .