

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 | 11/11/2024 | Jacobb Foster | Filled bracketed information, removed bracketed prompt and wrote my own response. |

**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 needs a web-based game called Draw It or Lose It that is accessible on multiple platforms. The game is currently accessible on android app compatible devices; therefore this software will allow users to access the client’s game on more platforms. Inspired by the 80s game show Win, Lose or Draw, however the teams will instead of a drawing guess from a rendering of a stock image. Four rounds, one minute a piece and if the guessing team does not guess after 30 seconds, other teams may steal until 15 seconds are up.

## Requirements

The Gaming Room’s Business requirements are that the application supports users on multiple platforms, the game must incorporate the spin-off rule changes. The software development environment should be set up for the Client, as well as a prepared software design document.

The Gaming Room’s Technical requirements:

* Teams must have multiple players
* One or more teams
* All team and player names unique for lookup
* One game instance at a time in memory
* Unique identifiers for games, teams and players.

## [Design Constraints](#_2et92p0)

* Designing a scalable and secure system
  + Game supports multiple games, teams and players. Leading to potential overflow if not accounted for.
* Cross platform compatibility
  + Client requests cross platform compatibility which adds difficulty when programming because developers must be mindful of compatibility.
* One game instance at a time
  + Utilize singleton class pattern to enable singular game instancing.

## [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 Gaming Room UML Diagram contains a package called com.gamingroom. This package contains the UML data that dictates the program functionality in the form of a diagram. Immediately there is the ProgramDriver, containing the main() function that uses the SingletonTester class to call testSingleton(). GameService class contains private attributes for a list of games, variables for the following GameId, PlayerId and TeamId respectively. As well an instance attribute. Followed by a private operation for instancing, but also public operations to get GameService, addGame(name) to add, getGame(id) and getGame(name) to search, then getGameCount(), getNextPlayerId() and getNextTeamId(). GameService(), Game(), Team and Player all have a 0..to many cardinality. Game() class contains a private teams() attribute to hold a list of teams. Then three public operations, game(id:, name:) for unique id and name. addTeam(name:) function that will create a team and add to the game. Then the toString() function to output the game data as plain text to the user. Team() class to create a team containing private attribute players(list) to hold a list of players for a team. Then three public operations for constructing a Team(Id, Name), addPlayer(name) to add a player to the team, then once again the toString() function. Player() class takes an Id and name as a public operation then a public operation toString().

**"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 is unix-based therefore allows for great performance and strong security. However will be higher cost and fewer cloud options. | Linux being open-sourced, stable and flexible enables support from a range of server configs and is compatible with many web hosting platform. But requires technical operating system knowledge to operate. | Windows is highly compatible with .NET apps and is highly user friendly. Although can be pricy to license and some consider it less secure than linux if setup improperly. | Mobile devices enable capability like location-based services and push notifications but lack in the processing power and storage aspect. This is not the ideal main server host. |
| **Client Side** | Apple upholds a high standard for its’ customers. Mac users will expect supreme graphics and smooth, seamless interaction. This could increase cost. | Linux operating system caters to more technical users so building in browser will be great. Linux has plenty of open source alternatives therefore development costs will be cheaper. | Windows is easily the most popular platform, so compatibility is going to be essential. Working with windows users’ primary browsers and ensuring compatibility will be the most important. | User experience will be the top priority when it comes to mobile development. Web-based applications on mobile devices typically can run poorly, leading to deterring users. Emphasis on UX is critical for mobile platforms. |
| **Development Tools** | Native macOS will use Xcode, Swift language and should consider a software like Unity for development. | Linux has a versatile selection of languages and software but Python and Java can be used as well as Visual Studio Code, or Vim. Docker and Kubernetes are supported on Linux. | Windows can use Visual Studio Code and has a wide variety of support for web based applications, as well as .NET suite. | Mobile devices typically will use Android or iOS, Android development and Xcode for iOS respectively. Unity can be used for testing and there are device compatibility software like Appium. |

## 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**: Linux is a cost-effective, open-source, compatible and secure operating platform for many web technologies.
2. **Operating Systems Architectures**: Linux is compatible with most modern programming languages like Python, Java and Ruby are excellent robust, compatible choices. As well as the diverse selection of IDE(s) to choose from. Visual Studio Code, Sublime Text, GNU Nano, Vim and Emacs to name a few great choices of development environments. Linux also makes testing easy with tools like Docker, Kubernetes and Jenkins.
3. **Storage Management**: Data storage can be quite costly and scaling solutions to existing physical data storage systems are even more expensive. Cloud-based storage solutions like Amazon Web Services, Google Cloud Platform or Hostinger. While this will likely be a hefty expense up-front their scaling solutions can’t beat beaten with a traditional home-server set up and scaling from there.
4. **Memory Management**: Redis is for memory caching and is especially great at handling complex data sets without overloading memory. This will meet the software requirement of one game instance in the memory at a time.
5. **Distributed Systems and Networks**: The combination of paging, segmentation, memory compaction and proper protection mechanisms. The game can uphold a reliable and efficient memory usage across all distributed systems. Using paging and segmentation we can reduce internal fragmentation and address external fragmentation. By utilizing TLBs, Translation lookaside buffers, we can minimize latency by providing efficient address translation.
6. **Security**: While typical domain hosting security rules will apply here, as the game should be hosted in a browser. There are several security implementations that should be factored. Data integrity and management is an essential security measure that should be accounted for by something such as TLS/SSL for data encryption. Additionally, The Game Room could consider an account for the user to have a username and password that’s attached to an email sign up. Adding even more user security as well as avenues for expansion within the newfound player base.