

Draw it or Lose it

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_l6ti7uoag22u)1

[**Table of Contents**](#_30j0zll)2

[**Document Revision History**](#_grjogdjh5fi8)2

[**Executive Summary**](#_sbfa50wo7nsh)3

[**Design Constraints**](#_2et92p0)3

[**System Architecture View**](#_ilbxbyevv6b6)3

[**Domain Model**](#_8h2ehzxfam4o)3

[**Evaluation**](#_2o15spng8stw)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 01/20/2022 | Christopher Williams | Initial creation of game |

**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 staff at the Gaming Room wish to make a web-based version of their Android game, “Draw it or Lose it” but do not know how to set up the environment. They need the game to accommodate multiple players per team, with multiple teams per game. There should only exist one game at a time. One solution is to hire an outside developer to develop the game so that it can be used across platforms.

## [Design Constraints](#_2et92p0)

The design constraints are that the game must accommodate multiple players per team. Each game must have multiple teams. There can only be one game in existence. Each player and team must have a unique name. This will require the implementation of a few design patterns, like the Iterator and the Singleton pattern.

## [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)

“ProgramDriver” is the class that contains the main method. It uses the class “SingletonTester” to perform the method, “testSingleton”. Using this method in ProgramDriver is a demonstration of Encapsulation, because the method is defined seperately and does not need to be redefined. It is its own package.

The “Game”, “Team”, and “Player” classes all inherit from the “Entity” class. This is Inheritance, meaning that they (and all their children), can use all the public methods and objects of Entity. Entity, however, is just an abstract class, meaning it is not used to perform any function on its own, but is more of a framework for building other classes. This is Abstraction, meaning that the user does not need to know about Entity’s “id” or “name” objects to use the player class. Entity handles those variables without any other need for input.

Game, Team, and Player all override the “toString” method to turn one of their instances into a string. This is an example of Runtime Polymorphism. The method is overriden, so it is assuming a different form, but it is still the same method. Game has a method to add a team and Team has a method to add a player. “GameService” is a child of Game and contains methods to create a game and to get the next player or teams ID. There can be many players per team, many teams per game, and many games per GameService. There can only be one GameService.

"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)

| **Dev Req** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac requires more expensive hardware and proprietary software. You cannot update the server app yourself. It is generally considered secure. Quick and easy to learn. Requirements for admin expertise are lower for Mac servers than any other OS. The command line can be used in configuration. | Can be run on any brand of hardware. Linux servers are lightweight and secure. All Linux products are open-source and free. The command line can be used in server configuration. The majority of servers are Linux. Linux servers require the highest amount of admin expertise. Changes can be made without rebooting the server. | Changes can not be made without rebooting the server. Windows is more vulnerable to security threats than any other OS. Software is proprietary. Has the widest variety of software options and plugins available. | Mobile devices often have limited data and upgrading hardware can be difficult. Software is open-source. It would be difficult to scale a mobile server to meet the demands of thousands of users on a data intensive application. |
| **Client Side** | Developing for Mac users is generally considered easiest on a Mac Server. Testing for Mac clients will require buying Mac hardware. Mac clients rarely introduce security vulnerabilities. | Linux comes in many different distributions. Targeting different clients can require a great deal of talent and time. Linux clients are considered the most secure. | Moderate expertise is required to develop the client-side of an application for Windows users. It is the most common OS, especially when it comes to desktop gaming, so this development is necessary. Windows users are the most likely to introduce security issues. | To fully support mobile customers, developers will have to create a mobile version of the site. Numerous front-end details must be changed from the desk top version. This could add up to a lot of time and money. |
| **Development Tools** | Tools required for this project will include Eclipse, Maven, Dropwizard, Java and the Terminal. Mac OS Server can be used for server construction. There will be licensing costs, but no need for a separate team. | Tools required for this project will include Eclipse, Maven, Dropwizard, Java and the Terminal. Apache, MySQL, and PHP can be used for server construction. Ubuntu Server is a custom OS for hosting a server. There will be no licensing costs, but there might be a need for a dedicated server development team. | The full version of Visual Studio is only available for Windows. It is one of the most popular and most functional IDEs. PowerShell and .NET are also only available on Windows. There are licensing costs and a separate team for server development is probably unnecessary. | Android Studio is used for mobile development. SQLite, Swift, Java, C#, and Kotlin are all commonly used for development. Purchasing a cloud server may be necessary if the application is heavily trafficked. |

## Recommendations

1. **Operating Platform**: The choice of operating platform is somewhat subjective, since there are numerous Operating Systems that have all the required functionality. My personal recommendation is the Ubuntu Server OS.
2. **Operating Systems Architectures**: Ubuntu Server has the ability to run on almost any hardware. It can be downloaded and installed for free, but enterprise level support is also available for a price. It has Docker, which is a container deployment platform that is fairly easy to use. Ubuntu Server also has Snap packages, which are a packaging and deployment system that works on many different Linux distributions. For a Linux system, Ubuntu server is considered very easy to set up and use. A specialized server development team may not be necessary.
3. **Storage Management**: The Gaming Rooms application is not very data intensive and will likely not support millions of players. The program is relatively simple and will not require a very high level of performance. For these reasons, I recommend using a cloud storage option. A cloud service provider will scale up with the demand placed on the servers for storage. It will also avoid the pitfall of buying hardware that quickly becomes outdated and obsolete. If the program is made more data intensive or draws a larger player base, then the Gaming Room may switch to in-house storage later. The Gaming Room should be aware that there are cloud options for ephemeral (aka temporary) storage and persistent (aka permanent) storage. For example, the images will most likely be stored on the cloud as a persistent object storage, which is not accessed as quickly as most other types of storage, but that is fine because it will only be accessed at the start of the program. After the images are transferred to the users device, it will remain in their memory until it is wiped out.

Cloud storage is fined per gigabyte, per month. Different types of storage from different providers have different prices. For example, object storage is usually cheaper than a more accessible, fast-access form of storage (like block or file) that is tied directly to the server. This block and file storage is where a database of player information would be stored, probably as a JSON file.

1. **Memory Management**: Linux uses virtual memory for the security and controlled sharing of information between processes. Every memory access has a virtual address that can be decoded by the CPU so that it will then point to a physical address. When the data needs to be placed into memory it is usually read from files and placed in a page cache to avoid having to perform disk access. If the kernel cannot find enough memory to continue operating, then the OOM Killer is invoked. This will choose a task to sacrifice in the hope that it will create enough space. However, most memory for this application will be either assigned to a database or immediately discarded.

All the data that will be created by this game can be stored in the cloud or on a Linux server database. Data that is needed for long-term storage will be stored this way. Any other memory needs will be handled by the server, then discarded. The short-term memory is discarded because it is unnecessary in a Restful API.

1. **Distributed Systems and Networks**: The server that hosts Draw It or Lose It will not be the only machine responsible for processing and data storage. A users local machine will have to carry some of the burden by performing operations like graphics rendering and interpreting user input. The users machine will also handle the volatile storage involved in the real-time processing of the game. The cloud or server database will store most of the data that is not volatile. Any requests for information from the server or database will be sent to the server or a proxy for the server as HTTP requests. The functions that will be hosted on a users local machine should be part of a download. The packages that need to be downloaded can be different for each operating system.

To be able to exchange data and commands with numerous operating systems, our server may consider using a proxy. Besides security, a proxy may be useful for routing client requests so that they can be handled in the appropriate order by the server. If requests are stacked in a queve in the proxy, then sorted through to determine which require processing and which can be ignored, then the server may avoid getting flooded with concurrent requests and improper requests that the server cannot or should not handle. In this case, an outage of the database would result in the users not being able to access or store information like player name, password confirmation, and game download. An outage of the server would result in the users not being able to access any online component of the game. The only thing they could do would be to access the information downloaded on their local machine.

1. **Security**: If the Gaming Room chooses to store data on the cloud, then they must research the security protocols and record of the cloud provider. Data should all be stored in one place to minimize the area of attack. Access to sensitive data and functions should require a user to have special access. Different domains should be established alongside methods for verifying that a user has access to that domain. Information could be encrypted and shared with users via a mutually shared key. Ubuntu Server may be more susceptible to personal attacks, since it is open source, but it is less susceptible to viruses and worms, since it is an uncommon OS. Using a proxy may hide the IP address of the server. The proxy may act as a firewall, denying any request that isn’t package a certain way, but it may still allow spoofing and tunneling. To avoid this hazard, two-factor authentication is recommended for any sensitive material access points. One use tokens and security questions would both work well for the second authentication factor.