

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 | 02/16/22 | Juan Pablo Izquierdo | Initial software design document |

**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 runs a game called Draw It or Lose it in the Android app store and is looking expand by developing a web-based game version of the game for multiple platforms. We expect to accomplish this by programming the game with design patterns which can be adapted to different platforms with a few changes.

## [Design Constraints](#_2et92p0)

Each new platform will require a different programming language, for example, SWIFT for iOS and C# for windows operating systems.

The game must have internet access to allow images to be downloaded for the game to render.

Multiple players must be able to connect to the same game, interact with the game quickly and simultaneously. The game highly depends on a quick connection because players need to analyze an image that is steadily rendered and enter a guess within 15 to 30 seconds.

A game can have multiple teams, and a team can have multiple players.

Only one instance of the game is allowed to run in each individual platform.

Games, teams, and players must have unique names or identifiers.

## [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 run the main method, starting the whole program. It also initializes game data by creating game instances.

SingletonTester tests that the singleton pattern utilized in the Games by checking that no games with identical names or ids were created.

GameService creates a singleton object that holds and tracks a list of games that have been created by assigning a unique id and name to each game, preventing duplicate id or name games from being created.

Game holds and tracks a list of teams that have been created by assigning a unique id and name to each team, preventing duplicate id or name teams from being created.

Team holds and tracks a list of players that have been created by assigning a unique id and name to each player, preventing duplicate id or name players from being created.

Player creates new players and prints out players.

Entity is parent class for Game, Team, and Player, which holds the id and name attributes and getter methods for the id and name attributes.

**"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** | It has many commands that add configuration flexibility. | Most popular for hosting webservers because it is flexible, the operating system runs very lean, and is also cheap. | It has a wide range of software available because of its popularity. | Mobile devices do not have the reliability of a server on any other OS. |
| **Client Side** | It has an easy-to-use interface, and most software very stable, however, it is pricier to develop for compared to other operating systems. | Linux has the lowest operating cost, but it is the least popular because most of the advanced functions of Linux cater to advanced users. | Many people and businesses use Windows because it is cheaper to maintain the current systems and changing them can be expensive or cause compatibility issues. | Portable and the most versatile out of all other operating systems. Programs must be optimized to run in many platforms with lower specs than desktops. |
| **Development Tools** | Programming languages: Swift, Objective-C, C#, Python.  IDE: Xcode, AppCode, Atom. | Programming languages: C, Perl, Java, Google Go.  IDE: Visual Studio, Sublime Text, Eclipse, Atom, NetBeans. | Programming languages: C++, Java, Python, C#, C, JavaScript.  IDE: Eclipse, NetBeans, PyCharm, Visual Studio | Program languages: Java, C#, Kotlin, C++, Python, JavaScript.  IDE: Android Studio, IntelliJ IDEA, Apache Cordova, NetBeans, Komodo. |

## 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 utilizing a Linux based server because they are the most cost effective and lowest cost for entry and maintenance due to some being open source. The light weight yet highly reliable Linux platform is ideal for server applications, and its support and compatibility with many other platforms is continuously growing in popularity. The advanced features that Linux often serve as a turnoff for entry level users should not be a problem because this will be primarily only concerned with the server side and its implementation should be seamless for client-side users.
2. **Operating Systems Architectures**: Linux has a simple architecture consisting of the hardware, kernel, shell, and lastly the applications that run on the platform. The hardware is the central processing unit (CPU), along with all other Input/output devices such as the monitor, keyboard, mouse, and memory and storage devices like solid state drive or hard disk drives. The kernel is the core of the operating system, among its tasks is providing management between the hardware, memory, and CPU. The shell is the actual platform in which programs run.
3. **Storage Management**: I recommend that a cloud storage system be used, since the data stored is not very large and the client only needs to retrieve one image at a time, this is the most dynamic way of handling storage for the game. The way this would be accomplished is by moving data from the servers to the client’s computer, then it can be sent from storage to main memory for the CPU to access. It can also be cached to be accesses quicker and more efficiently.
4. **Memory Management**: Memory management is primarily done with hardware, but several methods like contiguous allocation, paging, segmentation, and combinations of paging and segmentation can be used to manage it.
5. **Distributed Systems and Networks**: Linux support many other operating environments via shared-memory method and message-passing methods. Shared-memory requires that some variables are shared among processes and Message-passing allows processes to exchange information. These concepts can be utilized to setup communication channels for the game.
6. **Security**: Data must also be controlled so that it is only accessed by the authorized processes and users. The system must protect itself from unauthorized process access between users and errors to ensure proper execution. Parent processes must maintain control of child processes to prevent them from executing after the parent process ends or utilize more resources than allotted by the parent process. Tokens can be provided to authorized users to access data from the servers as required.