

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  2.0  3.0 | <4/22/2025> | <Opal Singer>  <Opal Singer>  <Opal Singer> | <First Edition, first section filled out>  Remainder of forms completed, including requirements and recommendations of proper system architecture. |

**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)

<Write a summary to introduce the software design problem and present a solution. Be sure to provide the client with any critical information they must know in order to proceed with the process you are proposing.>

The current problem to address is the migration of the software/ game “Draw It or Lose It” from Android to a web based system. Currently we are at the stage of selecting a proper web based system to transfer the software onto. The game must host servers that multiple people can gain access to simultaneously. Our top priority is creating a base system in which users can join a game through service, having that game run through the necessary steps, and allowing proper input and output to the players.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

Requires a number of players to be able to join a unique game session. Those user ID's must be unique. The players must have a method of input that allows them to geuss words and phrases. The system must in turn process this input and out the correct information from server to the client as the game updates.

## [Design Constraints](#_2et92p0)

Since the game is being transferred to a web based system, it must be able to run inside of a web browser or other easily compatible software. The Player needs a method of input for their answers. Given that the app must share properties of the original, it can deviate to far from the original design, less it be considered it’s own game separate from the original.

## [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.

On starting a game session, the server must create a local game object that is stored into the memory. Alongside this a process should be created in order to handle the instructions from the game. Either each player will act as a separate I/O device where the system determines the order in which the input is taken in, or all I/O will be done by a single handler, where the game will take in input from a single queue in the form of a component. The process will then handle this input alongside the instructions taken from physical storage and run in in the queue of instructions alongside input and output instructions. Periodically the game may take important information from the game, encode it to a readable format, then store it for later use. The process then continues until it is terminated.

## [Domain Model](#_8h2ehzxfam4o)

<Describe the UML class diagram provided below. Explain how the classes relate to each other. Identify any object-oriented programming principles that are demonstrated in the diagram and how they are used to fulfill the software requirements efficiently.>

"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.

Currently The first part of The UML Diagram shows a program driver creating an instance of a singleton object called, “ singletonTester”. It uses this class.

The entity object is an abstract class that the “Game”, “Team”, and “Player” classes derive from. Because each of them derives from the entity class, they have all the necessary variable in order to make sure that single instances of them can be held and checked through their inherited properties ID and name. The Game service Acts as a singleton, which aggregates a list of games. The numerical id on the association shows that each game service holds reference to somewhere between 0 and infinite amount of Game . The Game Object aggregates a list of Team in the same way. Then the team holds a number of players between 0 and infinity.

## [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** | The extended time to acquire proper licensing and technology to proper Mac servers would make the options look less than favorable. Not mentioning the higher cost of hardware for servers. | The open nature of linux system is promising for entry costs. The services will most likely be up the fastest with a linux based system. The lack of frequent security updates makes it a security threat without proper in house maintenance. | Windows tends to offer a balanced middle ground between ease of development and security. Frequent security updates allow for less concern of data leaks.  -cons, frequent security updates means our system can become outdated and become dangerous without adapting to new systems. | Our servers wont be running on an android system as far as i am aware, but instead interfacing with mobile devices. |
| **Client Side** | Client side the number of games played on macintosh systems is marginally lower. Otherwise considering the web based system, should not run especially different than other systems. | The somewhat manual nature of Linux systems may require the client to install certain software and drivers before interacting with our servers/ game. | Windows will allow the easiest means to connect to the game servers and should offer the easiest means to install drivers or software required for gameplay. | Mobile devices require an application or some built in systems to run the game properly. Assuming that the original game is built native to the android system. It will have to continue to develop with the ADK. |
| **Development Tools** | Obtaining the proper licensing to develop for a macintosh system is a lengthy expensive process. If it can be done through a type of virtual machine instead of natively, that should be considered first. | Development tools are openly available to Linux developers. The nature of it’s open source libraries will make it easier to develop, though may require developers that are used to working in Linux. | Depending on the system chosen to port the game, window the largest compromise between open libraries and secure systems. Creating virtual machines to test software is expensive but doable. | The development tools for android are easily accessible, very well documented and easily accommodate requirements of the software. There are built in systems that allow for virtualization of nearly all android devices supported by the company that creates the ADK.  This is less true for Apple products. |

## 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**: Windows, Largest customer base for gaming applications.
2. **Operating Systems Architectures**: A java based architecture would make transitioning from the original android application easiest and would possibly allow cross interaction between applications.
3. **Storage Management**: Offsite storage would best fit the company needs, minimal storage would be needed for game. The Game including it’s art and code would be stored server side, alongside recorded information of players and stored games. As far as the technical details of types of storage, large capacity disk storage would most likely be the correct choice for keeping costs down.
4. **Memory Management**: The memory would store the main process running the game services, new threads or processor components can be created by the server to run multiple game processes simultaneously. Either clusters of game can be groups together to run on a single thread or each game given its own thread. Since the main issue of memory comes down the storage of active pictures and renders, optimizing their format would allow us to cut back on memory considerably
5. **Distributed Systems and Networks**:   
   With a single system that connects multiple children, there are a number of ways to handle the communication between server and client. Using remote procedure calls could help with a number of issues such as different machines using different data types. By using an external data representation, we could more easily implement web applications for different devices without having to rewrite the communications protocols. As for the actuals servers and possible disruptions, if we are keeping our own servers to store data and run the game, it would be imperative to split the system into two pieces, one that runs the game and another which backs up the necessary data that the game stores. Doubling the storage side of the system allows for recovery of important files in case of outages and other accidents, whereas the main system is less at risk of accidents as it only puts the game service out for the time that it loses power.
6. **Security**: Most user input does not need high levels of security as it is simple data that it used to interact with the game. Implementing a basic security feature and hiding user data such as username and password behind an encrypted field are necessary.