

<Draw It of 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)4

[**Recommendations**](#_m8aleynsvzvc)6

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | <01/22/22> | <Mathew Denison> | <Updates to Executive Summary, Design Constraints, and Domain Model> |

**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 team at Gaming Room are looking for our team to develop a web application based game. The web app needs to be able to run on multiple different platforms. The game will be similar to Win Lose or Draw from the 1980s and will be named Draw It or Lose It. Instead of the players drawing the image, images will be loaded from a large library that will be rendered for the users. The players will compete to guess what is rendered in 4 one minute rounds. The rendering will happen over 30 seconds at a constant rate. If the team does not guess the puzzle before time expires, the remaining teams have an opportunity to offer one guess each to solve the puzzle with a 15-second time limit. This game currently already exists on the Android platform.

## [Design Constraints](#_2et92p0)

The design constraints are listed below:

* There needs to be greater than one team to play this game. Without a second team the game competition cannot happen, and the application of the other team guessing to “steal” would not be possible.
* Each team must consist of more than one person, as there needs to be multiple individuals guessing on each team.
* There needs to be a system to check if there is already a team name that is taken. Without this there will be issues with evaluation of who is on what team/win validation/potentially a list of other errors
* The games must exist each with their own number/identifier. Without this it could break the system of win validation and other potential errors.
* The game must run on multiple operating systems, not just android. This means that the application must successfully run on Macs, Windows PCs, and Linux machines. The main issue will be how to determine if we should change the language for each operating system (such as Swift for Mac) or if we can utilize multiple languages to enhance stability of code.

## [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)

Given the UML diagram, we find that Game, Team, and Player all inherit (an example of inheritance) from Entity class. In this example the objects they are inheriting are “id” and “name” which each of the given classes contain. We also see an aggregation relationship between Game and Team, Team and Player, and GameService and Game. We can see this given that each latter class has an instance of the former.

**"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** | - This does support server based deployment  - The up front cost for client would be most expensive due to hardware  - Tends to not be the OS used for server implementation  - Possibly less scalable  -Would have to write lots of code in Swift | - This does support server based deployment  - Up front cost is the least as it is free OS that is open source  - Also one of the most efficient Oss so server can be run on cheaper hardware  - Highly scalable and utilized in lots of server setups  - Supports many different coding languages  - Potentially least secure | - This does support server based deployment  - Up front cost would be less so due to mass adoption of OS  - Pretty efficient and has hosted servers for a long time  - Very secure and can utilize many different antivirus programs  - Supports language diversity | - Likely provides the least flexibility in hardware  - Not optimized for Server hosting  - Does not have as much support for server hosting  - Least cost efficient due to cost of chips that aren’t meant for hosting servers  - Possibly scalable for mobile platform hosting as code could be written in same xcode language as client side |
| **Client Side** | - This would take more time as Mac is a lesser used program  - It is also the most expensive hardware-wise for testing  - Expertise would require users to know Swift which may not be as common for developers | - Least up front cost due to cheapest hardware requirements for testing  - No cost for OS  - Not as widely used so may not be as accessible up front to developers  - It does require more knowledge on how to keep application up to date due to lots of OS updates | - Up front cost would be low due to high adoption of OS on hardware  - Most widely used OS for PCs so client will have more information on this  - Not as language agnostic which provides flexibility  - Less need for major changes due to less OS major OS updates  - More hardware variety for clients may cause issues with supporting the product | - Development kits would cost some up front  - Extremely widely used  - Likely would gain the most revenue due to higher adoption/utilization rates among users  - Support may be difficult due to hardware variety on Android but is opposite on iPhone  - Likely get the most out of hardware on user’s mobile devices |
| **Development Tools** | - IDE  - X-code  - Swift  - Object C  - Higher licensing costs and lower flexibility for different IDEs | - Eclipse/other IDE  - C language variety  - No licensing costs although less IDE support.  - Likely the least cost to get up and running | - IDE likely VS  - Coded in Visual basic or other popular language  - Great IDE support for many different tools  - Licensing costs involved in using VS | - Android Studio/Swift  - X-code  - Apple/Android development kits  - Java  - There are licensing costs but likely highest upside potential monetarily  - IDE support is based on what Mac is offered for iPhone and have flexibility for Android on Windows/Linux |

## 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**: My recommendation for The Gaming Room would be to use Windows as an operating platform to expand their total OS reach. Windows is one of the most widely recognized OS for computers and the majority of those who write code use Windows as their OS. You have large flexibility with IDEs, storage structure, Cloud integration, and low barrier to entry. From here you can expand your presence with developers writing native code on Mac and possibly Linux at a future date.
2. **Operating Systems Architectures**: The primary interaction that Windows uses for interaction with the OS and hardware would be through a Graphical User Interface. This is the graphical interpretation, input and output that user’s access (same as a shell) to make calls to applications, run processes, etc. The benefits to using this GUI (Shell) would be that it can be used through server and individual devices meaning that it is scalable for hosting the application and writing the code without developers needing to understand/learn command line operations
3. **Storage Management**: Windows has a pre-built-in tool called Storage Sense that allows you to manage data and auto delete content from folders that are not regularly accessed. Actions such as clearing your Temp folder would help conserve space on hard drives. This can even be accessed when writing code for programs on Windows machines to keep storage low for users. Other options include management of storage through File Explorer and setup tools to place applications in certain folders to conserve space. Also, Windows supports external hard drives which is a cheap way to add additional storage to a machine that is accessible from one machine to another.
4. **Memory Management**: Windows supports both Sequential and Direct Access, with direct access having even been improved with Windows 11. The benefits to this are we can write and maintain code that is scalable for all devices by writing in direct access for photo libraries. Since this is the most efficient way to access large data that is not sequential in a library, we would have massive upside to loading times and better gameplay experience. From here we can replicate sequential access for items such as lists or other sequential data types for further improved speeds.
5. **Distributed Systems and Networks**: The plan for hosting on a distributed network would be to host all servers over cloud. The benefits to this would be it is scalable meaning that storage space would not be an issue. With locally hosted services (local servers) one of the main drawbacks would be having to increase storage multiple times over the lifetime of the game. With cloud we could simply integrate with GitHub to push changes to the main line of code, then send that to the server. If we needed to make changes to code on a local server, we would likely have to pull the server down for maintenance, update code, and then re-execute. If we were to run out of space, we would need to clone the code to another server for hosting, or take the server down for maintenance meaning that the game would not be accessible for a time. Also, there is lower setup and maintenance costs long term for IT maintenance, as AWS or Azure would do maintenance on their end versus local IT people handling issues as they arise.
6. **Security**: One of the best reasons for choosing Windows are the built-in security features for local and server nodes. The Windows Security tool is included for free and is an industry standard for our local side. On the server/network side we would want to make sure to encrypt data so that it cannot be accessed. This includes making sure to write private accessors in the code so that it is hidden from being read. Also, the splitting of client to server (cloud) would enable further encryption/communication between services. The more pieces that a hacker/bad actor need to get through the better.