

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 0.3

## 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 |
| --- | --- | --- | --- |
| 0.1 | 09/18/2021 | Stockton | Wrote the Executive Summary, Design Constraints and Domain Model sections |
| 0.2 | 10/02/2021 | Stockton | Wrote the Evaluation section |
| 0.3 | 10/16/2021 | Stockton | Wrote the Recommendation section |

**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 client, The Gaming Room, wants to create a web-based version of their game Draw It or Lose It, which is currently only available as an Android app. The client has asked us for help on developing the environment for the game.

## [Design Constraints](#_2et92p0)

* Works as a web-based application
* Allows for multiple unique games
* Each game allows for multiple unique teams
* Each team can contain multiple players

## [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 ProgramDriver contains the main() method and is the class that everything is run from. The Entity class is used as the parent class for the Player, Team, and Game classes, it contains all of the items that are used between those three child classes. The Player class is used to represent the a player of the game. The Team class represents a team from the game, it contains a list of none to many Player objects. The Game class represents each instance of a game and contains of list of none to many Team objects. The GameService is a used to manage all of the Game objects. GameService is Singleton and so there can only be one instance of 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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac is a very reliable and robust operating system, but it’s lack of scalability and compatibility could pose a problem. Could be used as a server, but there are better options. | Linux is extremely customizable and can be modified to fulfill countless roles. it offers a lot of control to the user, which could be seen as either good or bad. Good choice for a server. | Windows is the most prevalent and compatible OS for server use. Most users should be familiar with Windows, so it would take less time to learn. While not as open-ended as Linux, Windows has enough room for specialization for it to be a good server OS. | Mobile devices lack the hardware necessary for use as an efficient long-term server. That being said, their inherit mobility makes them useful for small, portable servers. |
| **Client Side** | Mac shouldn’t be difficult to serve. Mac’s relative lack of variability means that less time would need to be taken to test different environments. | Serving Linux is tricky since Linux users will vary considerably in OS distribution and hardware. Serving this audience would require someone familiar with Linux and take significant time testing both hardware and software configurations. | While not as consistent as Mac, Windows is a much more consistent OS than Linux. There are a few Windows versions we would need to test for and would need someone familiar with version differences. We would also need to test different hardware setups as we did with Linux | Depending on the brand, serving mobile devices can vary in difficulty. Apple devices are much more consistent than Android for example. But the most important thing to consider is that the method of user input is different on mobile devices. As such, a program would need to be able to recognize a mobile device and offer a different user interface for touch screen controls. |
| **Development Tools** | The main IDE for Mac is Xcode, which is developed by Apple. The main language for iOS development is Swift, also developed by Apple. | The IDE contained in most standard Linux repositories is Geany. The main languages for Linux development are Python, C, and C++. | The main IDE for Windows is Microsoft Visual Studio. The main languages for Windows development are C# and C++. | Operating Systems vary across mobile devices and so do the tools for them. Visual Studio works for development on most platforms and the main language used would then be C++. |

## 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 server. With this, there is plenty of room for expansion while also providing the compatibility and customization that Windows offers. I would only recommend using Linux for the server if you have something specific in mind.
2. **Operating Systems Architectures**: The architecture of the server shouldn’t need to be too specific. Any Windows OEM server setup should work.
3. **Storage Management**: The server would store the compiled code as well as all of the images that will be used. The images will be a part of the package the end-user installs and won’t be constantly retrieved from the server. Information about games, players, and teams would also be stored here.
4. **Memory Management**: The server’s memory would be used to connect and send information between users. It would also be used when performing calculations and validations on the information it received about ongoing games.
5. **Distributed Systems and Networks**: The server would need to have a reliable internet connection to relay information between users. Due to the amount of expected traffic, the connection would need a larger bandwidth. However, relatively little information is transferred in each game, so you wouldn’t need an incredibly fast connection.
6. **Security**: Information should be encrypted before being sent. The server should take advantage of the security provided with Windows to prevent attacks. The server should also be cleaned and backed up regularly to prevent data loss.