

Draw It or Lose It

# **CS 230 Project Software Design Document**

Version 1.2

## 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 | 09/19/2021 | Scott Vanderwilt | Initial design document |
| 1.1 | 10/02/2021 | Scott Vanderwilt | Added Requirements |
| 1.2 | 10/17/2021 | Scott Vanderwilt | Added Recommendations |

**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)

Develop current Android app Draw It or Lose It as a web-based game that serves multiple platforms.

## [Design Constraints](#_2et92p0)

* Games need to support multiple teams
* Teams need to support multiple games
* Game and Team names must be unique
* Game must be singleton structure- only one instance can exist at any given time
* Servers must be updated to handle extra bandwidth
* Design must be reactive to a multitude of screen sizes.

## [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 class has one method main() that uses the SingletonTester class that has one method testSingleton(). The Entity class has two private attributes id and name. It has 5 methods a private constructor without any parameters and a public constructor with id and name as parameter. It has two getter methods to access the attributes and one toString() method. The Game, Team, and Player classes all inherit from the Entity class. The player class has a constructor with id and name parameters and it overrides the toString() method. The Team class has an additional attribute players that is a List of Player objects. So, the Team class has a 0 to many relationship with the Player class. It also has a constructor with id and name parameters, a toString() method that overrides the Entity toString() method, and a public method addPlayer(name) to add a Player object to the players attribute. The Game class has an additional attribute teams that is a List of Team objects. So, the Game class has a 0 to many relationship with the Team class. It also has a constructor with name and id parameters, a toString() method that overrides the Entity toString(), and a public method addTeam(name) that adds a Team object to teams. The GameService class utilizes the singleton class structure and can only have one instance available. It has 5 private attributes. A list of Game objects called games, so GameService has 0 to many relationship with the Game class. NextGameId, NextPlayerId, and NextTeamId are all attributes used in creating unique names for each Game, Team, and Player objects created. The service attribute is the only GameService instance ever created at any time. The GameService class has a private constructor, a public method getInstance() that returns the service attribute, an addGame(name) method that adds a Game object to the games attribute, and 2 methods to get a game object from the List games by either name or id, a method to return the number of game objects in the games List, and getters for NextGameId, NextPlayerId, and NextTeamId.

**"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** | Macs usually have the best hardware and good software security, but high cost and not able to run some windows programs. | Customizable hardware and lowest cost option but without Mac and Microsoft’s big money backing. | More programs readily available for windows, but higher cost and lower security. The cost is usually cheaper than mac. | Not powerful enough for heavy traffic. Might be good enough to serve games to a select few opponents or teammates. Limited security. |
| **Client Side** | Need to program for iOS is developing a stand-alone project. Developing a Web app that works on a browser such as Firefox or Chrome may also be an option. | Not a lot of Linux users on the client side, so any money put into specifically Linux development might be harder to recoup. A web app that works on a browser such as Firefox or Chrome is also an option. | Windows machines vary quite a bit in power, so most machines might not be able to handle large processor demands. A web app that works on a browser such as Firefox or Chrome is also an option. | Need to develop for multiple screen sizes in both orientations. Mobile devices update software continuously, so software must be maintained a lot more frequently. While a web app is an option here, actually developing a app for the app store is the better option. |
| **Development Tools** | A lot of ready to use development tools both free and paid are available for Mac. XCode, Eclipse, and IntelliJ are examples. Macs are generally equipped with higher processing power to handle the programs, but they still might be slow. Developers of all languages can use a mac machine. | Not as many tools as Mac and Windows, but there are options such as Sublime, VS Code, and Seamonkey. Computers need higher processing power to support these IDE’s. Developers of most languages can use Linux. IOS programs may be difficult to code on non-Mac machines. | A lot of ready to use development tools both free and paid are available for Windows. Eclipse, Visual Studio, and IntelliJ are examples. Some powerful windows exclusive options, but usually more costly. These tools generally require higher end processing power. Developers of all languages can use Windows; however, iOS programming may require additional tools. | Tools on Windows and Mac machines exist to develop for mobile devices. Apple generally uses Swift, and Android uses Java, but more and more options are becoming available. Multiple IDE’s have setups geared towards mobile development but using a device to develop an application is not feasible except for testing. |

## Recommendations

1. **Operating Platform**: I recommend using the Mac operating platform. It has a higher cost associated with it, but the costs are worth the extra benefits such as developing for iOS much easier, and higher processing power. I recommend using the IntelliJ IDE for development.
2. **Operating Systems Architectures**: Mac OS is Unix based. The operating system manages the RAM automatically and utilized protected memory that ensures each application remains in its own memory space and does not affect other applications.
3. **Storage Management**: I recommend using a MySQL relational database. This database is popular among developers and free to use. Data should be stored in the cloud using a RAID 10 storage system. This will for the data to be protected and easier scaling up as the company grows.
4. **Memory Management**: The Mac operating platform will automatically utilize RAM and allocate specific RAM for each application for security purposes. Java development has automatic memory management for the server. The server should have 16 GB of RAM memory to ensure no lag during high traffic.
5. **Distributed Systems and Networks**: I recommend hosting the application in a cloud-based server. This will allow for the servers to be maintained and upgraded as use increases. The server should be programmed in Java. I recommend publishing the game as a web app for Mac, Windows, and Linux use, and specific development for mobile devices such as Android and Apple to be available on their app stores for downloading. The web app can be hosted by the same company hosting the server for easier communication. The web app should be programmed in JavaScript. I recommend using React Native to program both the Android and Apple devices. React Native will then create the necessary files for app placement on both iOS and Android app stores. The web app and the devices would all communicate and receive the same information from the server, so a web user can play against a mobile device user. A reliable cloud-based hosting company such as AWS or Google Cloud will have contingencies in place in case of outages and will be able to upscale if needed.
6. **Security**: I recommend a secure user authentication system that uses roles, such as admin, user, and guest to allow the minimal amount of access necessary for each role. I recommend storing the username and password in the database utilizing the best and most up to date practices such as storing a strong hash of the password. I also recommend using a third-party authentication such as google or Facebook to maintain account security. I recommend using sessions to allow for easier use by the user once they are authenticated but making sure to use a reasonable amount of time for the session to expire. The database should store known devices and prompt for extra authentication (text, email, third-party) for unknown devices.