

Draw It or Lose It

# **CS 230 Project Software Design**

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 | 09/19/2021 | Zachary Hartt | Provided information relating to project on the Executive Summary, Design Constraints, and Domain Model. |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room would like to broaden their success from their Android app game, Draw It or Lose It, to a web-based client. The game will have an online multiplayer experience with specified teams that work together to complete puzzles within a given time limit.

## [Design Constraints](#_2et92p0)

1. Software Developers with experience in web development.
2. Provide an experience similar to their Android app.
3. Multiplayer platform with multiple teams consisting of multiple players.
4. Unique verification system to handle only one instance of the game at a time.

## [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 Entity class is the base class which holds common attributes and behaviors.

The Player class inherits the Entity class and provides a constructor to create players with particular id’s and name’s.

The Team class inherits the Entity class and creates a list of players, utilizing the Player class from 0 to infinity. Provides a constructor to create teams with particular id’s and name’s. Allows for adding players to the game.

The Game class inherits the Entity class and creates a list of teams, utilizing the Team class from 0 to infinity. Provides a constructor to create games with particular id’s and name’s. Allows for adding teams to the game.

The GameService class utilizes the Game class from 0 to infinity games. It creates a list of ongoing games, as well as multiple useful variables relating to game, player, and team id’s.

The ProgramDriver class handles certain instances and runtime code. It uses the SingletonTester class to ensure singleton operations are being obeyed.

**"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** | - More expensive.  - Requires higher-end hardware.  - Updates and upgrades typically cost more.  - More secure and less vulnerable to hackers.  - More reliable when compared to counterparts. | - Typically the least expensive option.  - Does not require high-end hardware for the most part.  - Free updates from open-source contributions. | - More expensive.  - Requires higher-end hardware.  - Updates and upgrades typically cost more.  - Frequent updates cause uptime issues. | - Limited features and scalability.  - Majority of people own a smartphone.  - Typically a lower cost operation. |
| **Client Side** | - Less user base.  - Proprietary software requires additional time, training, and understanding to corroborate an understanding of.  - Typically will cost more. | - Least user base.  - Largely used as a hosting client, so many developers will have experience with this interface.  - Most cost effective. | - Largest user base.  - Used often as a hosting client, so many developers will have experience with this interface.  - Mid-range in terms of price, usually. | - Large user base.  - Mobile-specific apps require more training, however a dynamic web application could be integrated into a mobile app and utilized the same, just with less feature than a mobile app. |
| **Development Tools** | Xcode, Atom, Visual Studio Code. | Eclipse, Komodo, Bluefish. | Git, Visual Studio, Visual Studio Code, Eclipse. | Xcode, Visual Studio, Droidscript. |

## 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**: For this project I recommend Linux as a server operating platform.
2. **Operating Systems Architectures**: Linux provides a robust server system while being low-cost and efficient. A Linux server will ensure minimal downtime to allow users a great experience with Draw It or Lose It. Linux is also widely used and accepted in the industry as a server platform.
3. **Storage Management**: Due to the limited number of items being stored (mainly concerning images), we should not have to be concerned about the amount of storage capacity our units have. We can store the data on the server and cache it through to the client on an as-needed basis.
4. **Memory Management**: Utilizing stack memory we can load, cache, and garbage collect images and other data efficiently. Less active memory will need to be used, creating a more performance friendly experience. Linux is a very minimal operating platform and open-source; it is an efficient platform.
5. **Distributed Systems and Networks**: The program will have a back-end (Linux, server) and front-end (client) of distributed software. If outages occur there can be data back-ups that store the most recent data. We can also have back up configurations (servers) that will kick in if the main server goes down.
6. **Security**: Linux, being an open-source platform, tends to mitigate security threats as people collaborate to fix vulnerabilities. User information can be stored safely and secured with encryption where appropriate. Limited data would be sent to the client based on other users information, protecting other users data. Abstraction and obfuscation would likely help with security of information.