

Draw It or Lose It

# **CS 230 Project Software Design**

Version 1.3

## 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 |
| --- | --- | --- | --- |
| 1.0 | 05/30/2022 | James Polley | Wrote Executive Summary, Design Constraints, and Domain Model |
| 1.1 | 06/12/2022 | James Polley | Wrote Development Requirements |
| 1.2 | 06/19/2022 | James Polley | Evaluate different operating platforms |
| 1.3 | 06/26/2022 | James Polley | Wrote recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room is seeking to expand its existing Android app, Draw It or Lose It, onto more platforms. Users will be able to play on the same team with other users, and against other teams. The aim is to deliver a similar and fair game experience for users across different hardware and operating systems.

## [Design Constraints](#_2et92p0)

* Users across different operating platforms should have similar experiences engaging with their respective apps. Data and timing should be synchronized for all players in a game. This will especially be a challenge, given the distributed environment on which the apps will be run.
* Games should be able to have one or more teams, with multiple players assigned to each team, and each player limited to being on only one team.
* A single and centralized module will be needed to ensure game and team names are unique. This will prevent duplicate games and other issues related to players joining teams and games.

## [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 parent class to Game, Team, and Player. Since these three share the attributes of name and ID, there is less code duplication in each class by having them all inherit from Entity. Since the attributes are private, they cannot be altered once the object has been created. This contributes to ensuring IDs are unique.

There is a hierarchy in how objects can be created. A Player must be created by a Team, which must be created by a Game, which must be created by the GameService. Since each of these object types rely on each other necessarily, this dependency is appropriate.

Lastly, GameService’s Singleton pattern has been expanded to include keeping track of team and player IDs. By using accessor methods for private variables, the class ensures that data isn’t altered by external objects, other that indirectly by incrementing after accessing.

This model makes extensive use of the Object-Oriented Programming principles of encapsulation and inheritance.

**"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** | **Characteristics:**  Unix-like  Open source  GUI  **Advantages:**  Works well with Apple devices  Easy administration  Easy workload distribution on multiple machines  Good support  Unlimited user license  **Weaknesses:**  Only runs on Apple hardware  Fewer third-party applications | **Characteristics:**  Unix-like  Open source  GUI  **Advantages:**  Free  Highly secure  Wide distribution  **Weaknesses:**  More technical knowledge required  Difficult to navigate  Lack of long-term support | **Characteristics:**  Closed source  GUI  **Advantages:**  Many options  Good OS support  Good third-party application support  **Weaknesses:**  User-based licensing  Less secure | Doesn’t seem like a good idea to host an application server on a mobile device. Or maybe it’s a great and revolutionary idea… |
| **Client Side** | High/moderate cost  Some experience with MacOS is helpful  Less time needed to develop | Low cost  Much experience with Linux is needed  More time needed to develop | Moderate cost  Some experience with Windows is helpful  Less time needed to develop | Low cost (in addition to setting up on Mac, Linux, or Windows)  Some experience with iOS and Android is helpful  More time needed to develop (range of devices on iOS and Android) |
| **Development Tools** | Xcode | Visual Studio | Visual Studio | Xcode  Android Studio |

## Recommendations

1. **Operating Platform**: I recommend the Windows operating platform. It is common, so many developers are familiar with it. It has a massive ecosystem of available software that can assist in expanding and scaling the game.
2. **Operating Systems Architectures**: Window’s OS includes all the standard features such as file and memory management. It allows for multi-processing, which makes it ideal for running a server and/or using multicore processors, which are becoming increasingly common today. Lastly, it included tools for creating and rendering a graphical user interface with the intuitive concept of the “window”.
3. **Storage Management**: I recommend using Microsoft Azure cloud storage. When it comes to different storage service providers, all share the ability to scale seamlessly. Pricing is also comparable. By staying within Microsoft’s suite of products and services, it is likely that it will be easier to get support for any issues that arise.
4. **Memory Management**: Windows uses virtual memory management, which swaps data in and out of storage strategically in order to increase the volume of process able to be managed by the OS at any given time. This is especially useful when the amount of data to manage exceeds the hardware’s memory capacity, which will certainly be the case when running the server.
5. **Distributed Systems and Networks**: <Knowing that the client would like Draw It or Lose It to communicate between various platforms, explain how this may be accomplished with distributed software and the network that connects the devices. Consider the dependencies between the components within the distributed systems and networks (connectivity, outages, and so on).>
6. **Security**: Security can be assured by encrypting all traffic between client and server. This is accomplished by using https protocols. In addition, each API should require an authorization token. Lastly, the applications should be developed to exclude any functions that trust user input, as these create vulnerabilities. If the server is run by a cloud service provider, they will handle security of hardware and infrastructure.