

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc115077317)

[**Table of Contents 2**](#_Toc115077318)

[**Document Revision History 2**](#_Toc115077319)

[**Executive Summary 3**](#_Toc115077320)

[**Requirements 3**](#_Toc115077321)

[**Design Constraints 3**](#_Toc115077322)

[**System Architecture View 3**](#_Toc115077323)

[**Domain Model 3**](#_Toc115077324)

[**Evaluation 4**](#_Toc115077325)

[**Recommendations 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/12/23 | Christina Gray | Completed Executive Summary, Design Constraints, and Domain Model sections. |

**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)

## Based on the popular Android game, Draw It or Lose It, the Gaming Room needs a web-based game that can serve multiple platforms. A similar game show from the 1980s would be Win, Lose, or Draw.

## [Design Constraints](#_2et92p0)

- Game must be written in a web-based language such as JavaScript or Python in order to be fully

functional in any web-based environment.

- Must be written to allow multiple teams with multiple players per team by implementing team

and player objects.

- Must check game and team names to ensure uniqueness by using a singleton pattern in game

and team creation.

- Must allow only one instance of the game to exist in memory by creating a unique Id for each

instance of game, team, or player by using iterator patterns.

## [System Architecture View](#_ilbxbyevv6b6)

Each Game, Team, and Player class inherits from Entity, so they can access attributes and operations contained in Entity once and use them in their own classes. Games, GameServices, Teams, and Players are all interconnected. They use a zero-to-many associations, which means they can use as many instances of the other as they need or none at all. ProgramDriver drives the package by inheriting/using SingletonTester.

## [Domain Model](#_8h2ehzxfam4o)

Entity is our super class, and Game, Team, and Player inherit directly from it. There is a reference in the GameService to the Game class, which refers to the Team class, which refers to the Player class. Aggregation is the process by which these classes are linked to one another. Our SingletonTester and ProgramDriver classes are also included. ProgramDriver is where our application is executed, and it uses SingletonTester as its test class. There is also a relationship between SingletonTester and ProgramDriver. Multiple teams (from a list) can play at the same time with multiple players (from a list) thanks to the way these classes are set up.

**"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** | Mac OS X server can be used with Mac OS X. Apple's website states that Mac OS X Server costs only $20, so implementing it would be inexpensive. However, Mac is not as popular for performing these tasks as Linux or Windows. | There are many Linux distributions that have server capabilities, which makes Linux an interesting choice here. It is low-cost and open source (which provides a lot of resources). It would be helpful if you had someone who is familiar with Linux running the server, since many users aren't familiar with it. | Windows Server is offered by Windows. It might be costly to implement, but it is fully functional according to Microsoft's website. Because Windows is the most widely used operating system, it is likely that users can be found to operate Windows servers | Due to the limited power of mobile devices, hosting a fully-fledged server on one may not be the best option. Getting a server up and running on a mobile device is the most cost-effective option. |
| **Client Side** | Due to the fact that these operating systems are not open source, the cost would be similar to a Windows setup. An expert in Mac would require less time than someone who lacks experience in Mac, but someone who has little experience in Mac would require more time. | Cost would be low  (if there even is a  cost) with Linux,  as it is open  source. Maximum  time and  experience would  be necessary, as  Linux is not  commonly used  and you would  need someone  who is apt with  Linux and allow  them time to  work, as Linux can  be difficult even  for someone with  experience. | As these operating  systems are not open  source, so the cost  will likely be  expensive. Time  would rely on  expertise, so it would  take less time for  some who has  Windows experience  and more time for  someone who does  not have as much  Windows expertise. | Cost for mobile  devices should  not be too much  of a problem. As  mobile devices  may be easier to  deal with,  experience might  not be too much  of a concern.  More time would  be required, as it  would require  working on  multiple  operating  platforms and  multiple mobile  devices. |
| **Development Tools** | Swift would be the  more common  language used to  write applications  for Mac. There are  multiple IDEs that  can be used for  Swift, such as Atom. | Eclipse and Atom are commonly used IDEs on Linux. Java is the primary language supported by Eclipse, but other languages such as C+ can also be supported | Eclipse and Visual  Studio are popular  IDEs for Windows.  Visual Studio can  be used for  developing in  HTML, C# and  JavaScript among | Development tools for iPhones are similar to those for Mac, and iOS apps are typically written in Swift, although iOS and macOS have a few differences. |

## 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**: Based upon all my research, I believe that Windows would be the best operating platform to use, as it is the most widely used operating platform, as well as the one that most people have a working knowledge of. There is an abundance of IDEs that can be used with Windows, and the total cost to utilize it is typically lower.
2. **Operating Systems Architectures**: Application programs can utilize Windows' kernel processes without directly affecting them. Application developers can use Windows' power to set up GUIs and windows, access memory, and perform other essential functions without affecting the processes that make the operating system work.
3. **Storage Management**: As part of the Windows operating system, Disk Management and Storage Sense are included. There is also a Disk Cleanup tool available in Windows. Storage Sense is a Windows system utility that helps maintain the storage on the system by deleting unnecessary files that are taking up space. Disk Management is used for advanced storage tasks, while Disk Cleanup is used to remove unnecessary files.
4. **Memory Management**: System utilities such as Memory Management are built into Windows. The application would need to be able to access all the game's image files through a database.
5. **Distributed Systems and Networks**: Our game will be distributed via a client-server model, with each client app dependant upon the game's server app, so that each client app can be tailored to its individual system's capabilities. It is also necessary to have a strong server network since this game is dependent on the ability of multiple clients to access one server at the same time.
6. **Security**: In any Windows system, Windows Defender is a built-in security feature. It would be necessary to encrypt all data sent back and forth with tried and true encryption methods.