

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 | 03/23/25 | Dan Story | Added software design |
| 1.1 | 04/06/25 | Dan Story | Proofread and made minor edits |
| 1.2 | 04/20/25 | Dan Story | Added more details to 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)

The customer, Gaming Room, has an application called Draw it or Lose it. They want a web based game service that will work on multiple platforms based off the current game. The staff are having problems setting up a hosting environment for their game. They want the ability to have one or more teams involved, teams should have multiple players, and the game and team names should be unique. Only one instance of that game should exist at any given time.

## Requirements

-Multiple platforms

-instanced unique games

-one or more teams for each instance

-one or more players for each team

-games, teams, and players should have unqiue ID’s to prevent duplicates.

## [Design Constraints](#_2et92p0)

Each platform will have it’s own dev kits, so the application should be splashable, ideally for a desktop, android, and IOS application. Unique instances of the game puts a hardware limit on how many games can be run simultaneously. Players will need to be alerted if a unqiue team name has already been chosen/in use. There should only be one instance of a given game at a time so player’s can connect to the instance they prefer.

## [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)

ProgramDriver has the main method. It also uses the SingletonTester to check if there is an instance of GameService. The Game, Team, and Player class all in are children to the Entity class, and are extended from it. A Game Service has multiple Games, Games have multiple Teams, and Teams have multiple Players.

The Entity class uses inheritance to build it’s children  
Each of the children are encapsulated to keep their 0 .. \* relationship  
The Polymorphism let’s each Game have unique teams, and each player to have unique teams.

Abstraction comes from the only time the user see’s any of this is when letting a player/captain know that the team name already exists.

"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** | Apple licenses are generally more expensive, but are very easy to interface with each other due to them being built in the entirety of the Apple ecosystem. This can limit adaptability. | Open source with strong security, and great adaptable resource optimization, but may have problems being compatible with every device as well as a learning curve for use. | The middle ground between Mac and Linux. Still requires a license, but with more customization options. Windows is also not as optimized as a Linux server could be. | Mobile devices are not ideal as a server. They lack power, run off an unreliable battery, and are generally on a more unreliable network. |
| **Client Side** | Mac’s are generally a more closed ecosystem. It’s very easy to develop for multiple “types” of macs since they’re so similar, but they may lack options or tools due to the closed nature of their ecosystem. | Linux is generally more open source, but planning for every eventuality of a linux system can be hard. May need specific expertise from a user to make things work. | The middle ground between Linux and MacOS. May not be as secure as the previous two systems as well, but supports a wide range of software applications. | Split between two ecosystems, IOS and Android. IOS is a closed mac style ecosystem while Android is open source. The android store has a greater market share, but the apple store has greater quality control and more conformance to standards. |
| **Development Tools** | Xcode is Apple’s IDE. Supports swift, c++, obj-c, obj-c++, applescript, python, ruby, resedit, and C. Swift is the preferred language. | Has multiple IDE’s as well as command line tools and open-source dev tools. Can support any language, but may require special expertise. The low level control languages are preferred. | Has multiple IDE’s to choose from like visual studio, and is adaptable for many languages. C# is what Window’s prefers and microsoft’s IDE is Visual Studio. | For IOS, Swift is the preferred language, and Xcode is the preferred IDE. For Android, Google prefers Kotlin, Java, and C++ and the official IDE is Android Studio. |

## 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**: I would recommend using Unity. It has great game development flexibility, and is supported by Windows, Mac, Linux, Android, and IOS. It’s also supported and easy to port to major game consoles. Many games built in Unity have been ported to the Nintendo , Microsoft, and Sony Playstation stores.
2. **Operating Systems Architectures**: Unity uses a single code base that compiles natively for the target operating system. This lets you re-use your code for multiple platforms, including the aforementioned game consoles.
3. **Storage Management**: For Unity, it’s recommended to use their cloud based storage service or something trusted like Amazon Web Services (AWS). You can also use a local storage system, like solid state drives for faster access times.
4. **Memory Management**: Unity uses what it calls a “mark and sweep” algorithm. It traces all objects it can reach from the “root” (like global variables) and marks them as “alive”. Then it goes through everything (the heap) and reclaims anything not “alive” (such as variables no longer being used).
5. **Distributed Systems and Networks**: Utilizing the Unity Cloud service, Unity has a distributed authority service. This allows users to connect and use the already existing service to access their game. If the server has an outage, players can simply move to another server that is running the service. This would also let users who have a “stand alone” version of the game connect to the services. As long as Unity doesn’t have an outage, your game doesn’t either.
6. **Security**: Use a role based security system, this provides a tiered security system where if a team is compromised, the entire game isn’t compromised, and if a game is compromised the entire service isn’t compromised. This way you can restrict information as needed. You should also only escalate the least amount of permissions that users and administrators need. Unity itself also has an ongoing security team where they regularly release security patches as well as off “bug bounties”. If someone finds an exploit, unity will pay them to turn it over to them to fix.

*All relevant information about Unity comes from their own site at https://unity.com*