

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 | 07/13/2023 | Andre Freitas | Added Executive summary, requirements of project, design constraints, created system architecture view, domain model, wrote evaluation, gave recommendations. |
| 1.1 | 7/30/2023 | Andre Freitas | Added Server side, client side, and development tools to template |
| 1.2 | 8/11/2023 | Andre Freitas | Filled 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 problem that the client is having with the software is that the staff at The Gaming Room do not know how to set up the environment of Draw It or Lose It. We must keep in mind any hardware restrictions that may cause trouble for the project. The project must implement a time limit of 1 minute per round, 4 rounds, and a drawing must be completed at the 30 second mark.

## Requirements

The software requirements the team has include:

* The game must be able to have one or more teams.
* The teams must be able to have multiple players.
* Game and team names must be unique and alert players when they are not.
* Only a single instance of the game can exist at a time.

The business requirements are:

* The game must work on multiple devices and operating systems.

## [Design Constraints](#_2et92p0)

The game must work well between multiple OS and be able to communicate between the different users without anyone lagging behind and not having the opportunity to view the puzzle. The game must also be limited to a single instance and not allow any duplicates for Game or Team names. The programming language must also be able to work on several OS. The game originally worked on Android and must be able to be ported over and compatible with other OS.

## [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 driver class will be used to begin the creation of games, teams, and players. The singleton class of GameService is what actually creates the instance, and since it’s a singleton there can only be a single instance of it at a time. GameService sets it’s constructors as private so they cannot be affected by any other variables or classes. This keeps an instance of GameService from being started by any other method. The getInstance() method checks if GameService has started. If it has it will not start a second instance.

The driver class calls the method addGame() when GameService is running. addGame() does not allow multiple Gabe objects of the same name to exists. If it does not exist, it gets added to the List named “games”.

After a game is created teams can be added through the addTeam() method. addTeam() uses the same type of code as addGame() to prevent multiple team names from being added onto the list. If the name does not exist on the list then it is added on.

Game, Team, and Player are subclasses of Entity. Entity has two protected attributes named “id” and “name”. The default constructor is protected so null objects are blocked at the point of creation and only overloaded constructors can be used.

There are several Object-Oriented techniques used in the below UML. Polymorphism and inheritance are use in the Entity class with both the extension of it and with how we overload it’s constructors. Encapsulation is used to add teams as a team object cannot be created directly since the constructor is protected, but the use can still add teams through the addTeam() method.

**"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** | The OS X Server is available for the Mac but requires the client to purchase their own hardware. This may not be the most affordable or simplest option. | Linux, at the time of research, is the most popular webhosting OS due to cheaper maintenance and license costs compared to windows. This is due to Linux being opensourced rather than closed. | Windows are very familiar to most users and many of their applications are broadly used in multiple settings which makes them easier to use for most people. License costs however tend to be higher than other OS types. | Mobile devices are not equipped for serving multiple users despite being able to be used as a file server. Their hardware is much more limited, such as having less RAM. Costs are difficult to predict for Mobile devices as they would probably need to be made custom. |
| **Client Side** | Macs need to developed by the most recent version of Xcode on a Mac computer. They also are coded in lesser known languages and have a much smaller market compared to Windows OS. | Linus uses commonly used languages such as Java, C++, or Python which makes development easier.  Multi User support has little use however as Linux is developed on a GNU/Platform that is not commonly used. | Windows uses C# or .NET which are both common creating an easy development of a Windows client app. Windows has also been multi-user since Windows XP. Windows is also very widely used, making it better for business. | Mobile devices are not designed to be multi-user but design for client applications is not comparatively difficult. Android phones are Java based so windows and Linux can be used as a base for the code.  iOS iPhone and Mac share similar hardware needs as well. |
| **Development Tools** | Max uses SWIFT and Objective C languages.  xCode is the common IDE for Mac and is listed at $99 per year. | Linux development often uses C/C++, Java, or Python.  Python IDE’s are often free like NotePad++ or PyCharm.  C++ has numerous IDE’s, but not all are available for Linux. Eclipse is and is free. | Windows is developed mainly by C# and .NET. Microsoft Visual Studio is both popular, easy to use, and has many plugins built in.  The cost of Visual Studio can be from $45-$250 depending on features per year. | Android SDK is Java based and is used widely and Android Studio is free to download.  iOS’s languages, Objective-C and SWIFT, are almost exclusive in Xcode.  Xcode is listed as $99 per year. |

## 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**: Linus is the suggested solution for the game. Their servers have less cost and do not have the same limitations on their data centers that Windows servers do and can connect to the frontend via API.

Linux also offers good security and operability and many tools are made available, including security software. The front end will be agnostic and can be written in whatever is the preferred language for that platform, whether it’s iOS, JAVA, .NET or other.

1. **Operating Systems Architectures**: Backend servers that manage the game environment and frontend/client based rendering would be my suggestions. There’s no need to keep low latency in the game as fast reflexes are not a required part of playing it so a lower frame rate is no issue.

Frontend rendering allows the server to offload more resource intensive portions of The Gaming Room which would reduce the monthly costs. Client side rendering would keep framerate issues under control and could cache subsequent images on the client’s side to allow smooth transitions to new images.

1. **Storage Management**: SSD or HDD could work for storage along with client side rendering and some kind of caching behavior. Server-side, cloud tools will add flexibility for scaling or localization.
2. **Memory Management**: Linux uses pagecache for data stored in main memory virtual memory for any pages that it is allocated. Linus uses demand paging which places pages not currently being used outside the memory. This will cause lower memory usage.

The Andre Runtime use paging and memory-mapping to manage memory. This means that any memory an app modifies remains in RAM and cannot be paged out. iOS also allows us to avoid retaining and releasing objects as Xcode takes care of it in realtime.

Minimum RAM on the server will be needed with client-side rendering. This could increase costs however as it scales with number of users. Only 1-2 images need to be stored at any point so RAM requirements should be low.

1. **Distributed Systems and Networks**: Cloud native architectures help prevent uptime issues and outages, many providers of cloud software can replicate and shift services among deployments to avoid large scale outages. The frontend and backend both communicated through RESTful API which lets communications stay transparent between clients and servers.

**Security**: Security will be role-based. Entitlements need to be made to ensure users only have access to the minimal functions needed for them to preform what they need. These roles could be expanded in the future but will start very limited. We will follow the idea of least-privilege. No users will be an Admin on the system.

API’s will be encrypted and a firewall will be added using industry best practice as it’s settings.