

Draw it or Lose it

# **CS 230 Project Software Design Template**

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 | 11/14/2021 | Jordan Barnes | Initial architecture and design recommendations. We have our chief complaints and issues outlined within this initial submission. |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room wants to develop a web-based game that serves multiple platforms based on their current game, Draw It or Lose It, which is currently available in an Android app only. To facilitate the development of the web-based version of the gaming app, we need to address the software requirements, prepare a design document, develop the game application, and keep in mind all hardware requirements.

## [Design Constraints](#_2et92p0)

* A game will have one or more teams involved.
* Only one instance of the game can exist at any given time.
* Game and team names must be unique.
* Each team will have multiple players assigned to it.

The Gaming Room wants to have this run on all devices and not be limited to only android. With operating platforms like Windows, Linux, and Apple we need to either re-do the entire codebase in a language that can be interpreted by its respective platform or find a way to bridge the original codebase to each endpoint.

## [Domain Model](#_8h2ehzxfam4o)

Game, Team, and Player classes all inherit information from Entity. With this UML we can see the inheritance in effect. All of our classes will share common traits with Entity. We can also see that Gameservice has a reference of Games, Games a reference of Team, and Team a reference of Player. Meaning we are seeing a common aggregation and inheritance principle between each of these types and Entity.

**"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) summary

**Server side:**

Each operating platform offers a sever based deployment method where our website will be hosted. Licensing with Linux is open source and free of cost whereas any other tools or scaling mechanisms will be free of cost as well.

There aren’t many options for servers other than windows or Linux. We have outlined the advantages and disadvantages in the tables below for each.

**Client Side:**

**Windows** uses visual studio and visual basic as a default, its quite hard to develop network-oriented applications with visual basic but it is possible.

**Mac** uses XCode and swift, and applications are not intensive in nature of creation.

**Linux** uses eclipse (most popular) and C is its default language, its versatile and its complexity provides great security.

**Android** uses Android studio as its default and java. Its versatile and a overwhelmingly saturated market.

**IOS** uses XCode and Swift. Very interactable with Mac operating systems and just as versatile as Android.

**Keep in mind this is just a summary, more detail follows in the table below.**

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Advantages:  MacOS server offers good connect ability for other Mac clients on the network. There is an easy creation of features comparable to Win group policies for Mac clients. Mac is also popular for web hosting.  Disadvantages: Unfortunately for server hosting, and web hosting services MacOS is not very popular. | Advantages:  Web hosting services and platforms prefer Linux over the others. Security is much more streamlined and catch-dry than other platforms  Disadvantages:  Its difficult to find support as the platform is niche. | Advantages:  Windows dominates the market in usage and compatibility. Market is extremely saturated and easy to find web hosting.  Disadvantages:  Bad security. | Advantages:  Very portable and easy to use on the go. Cost is less than other platforms.  Disadvantages:  Awful security. |
| **Client Side** | Cost is similar to windows and shouldn’t create a large funding gap, expertise is average due to the saturation of Mac developers. We need to determine what is needed to make sure its compatible with all Mac platform devices/mobile devices. (Apple) | Since Linux is quite niche in its community this will be where we need most of our expertise. SDLC is going to have to be much more attentive than our other 3 platforms. We need to make sure like the others this will be compatible with our other platforms. | Cost and expertise for this end will be little to none comparatively. The market for windows-based developers is saturated. Again, we must make sure that our development process ensures compatibility. | This will be moderate to minimal difficulty. The market isn’t as saturated as windows but not as niche as Linux. We will just need to focus on compatibility. |
| **Development Tools** | Swift is the most popular language for MacOS and IOS. We can use any language we would like as Mac is compatible with nearly all languages and we are not limited to Swift. Backend and Frontend compatibility will not be a major issue with Mac. | Linux can work with most main-stream IDE’s and languages. There will not be a support issue with end to end compatibility as we have access to CSS/JS and Python, Ruby, Java. | Windows is far less complicated as Mac or Linux and we have main-stream access to all IDE’s and Languages that is available on Linux and Mac. We can use HTML, CSS, Ruby, JS, Python, C++, C# we also have access to VS as well. | Swift and android can be run on all three machines. We can use CSS/JS for our front end and Python/Ruby/Java for our Back end. End to end with our mobile platforms should not present any more issues than the other 3 platforms. |

## Recommendations

1. **Operating Platform**: We need to start on windows as it is saturated and the most popular. I don’t believe going with Linux would be a developmentally stable choice due to cost effectiveness.
2. **Operating Systems Architectures**: Windows will have a variety of applications that will assist us in developing UI, graphics, instancing, hosting, web services, etc. Developers in the system architecture vector will be easy to find as well.
3. **Storage Management**: We can offset storage capabilities with platforms like AWS and other windows compatible storage systems. We also have built in storage systems within Windows for large projects that we can back up so we don’t lose.
4. **Memory Management**: I believe it would be prudent to choose a multi paradigm language with cost effective memory management as we will need to store large amounts of data. We’ll have to be rendering and multithreading at runtime within a single instance of our game which will be memory intensive. Fortunately, windows has a fantastic memory management platform.
5. **Distributed Systems and Networks**: I did some googling to see if there is a way to export our game for different devices much like exporting a PDF as a PNG. I found that there are several software platforms that do this for us. The most popular being Develop 4. This gives us the ability to export as a web-based, Android, IOS, and other things that allow cross-compatibility.
6. **Security**: Windows has firewall and built-in antivirus protection as well as other security dams. We have embedded software to take care of these already, but I believe it would be prudent to outsource our security to a third-party software like Fortinet.