

Draw it or Lose it

# **CS 230 Project Software Design Template**

Version 1.2

## 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 | 03/20/22 | Roy Romero | First Complete Draft of Software Project |
| 1.1 | 4/3/22 | Roy Romero | Evaluation Revision |
| 1.2 | 4/17/22 | Roy Romero | Recommendation Revision |

## [Executive Summary](#_sbfa50wo7nsh)

Our new client *The Gaming Room* wants us to develop a web-based game that serves multiple platforms based on their current game, Draw It or Lose It. This game is similar to Win, Lose or Draw, where teams compete to guess what is being drawn. A single game consists of four rounds of play lasting one minute each. Drawings are rendered at a steady rate and are fully complete at the 30-second mark. If the team does not guess the puzzle before time expires, the remaining teams have an opportunity to offer one guess each to solve the puzzle with a 15-second time limit. The design problem is that the game is currently only available for android, but our client wants it to be multiplatform.

## [Design Constraints](#_2et92p0)

* Most importantly, game has to be multiplatform
* Only one instance of the game is allowed to exist at a time
* You need at least one team
* Game, Teams, and Players have their own unique identifiers

The main purpose our client contacted us is to make this game into a multi-platform application. We can make this game multiplatform on Windows, Linux and mobile devices by altering the code or adding to it to meet requirements for those platforms. When it comes to apple devices I suggest getting a separate team to translate the current code to Swift. This team can work side by side with the main team in order to get both versions of the game up and running in similar timeframes.

## [Domain Model](#_8h2ehzxfam4o)

Going through the UML diagram below you can see the newly created “Entity” class has parent relationship with the Game, Team and Player classes through inheritance. This way the unique identifiers (name, teams, game id) are shared across the entirety of the code. Making the Entity class a parent class and the Game/Team/Player classes child classes. We do this to minimize code on our Main function, make our code flexible in case of changes and have our program run as efficiently as possible.

**"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** | Advantages:   * Solid customer support * Synchronicity between all apple products * minimal virus attacks. * Does have server-based deployment method for host   Disadvantages:   * That they have minimal software * Cannot read any windows format. * Cost for putting game on apple app store but development tools are low cost | Advantages:   * Linux is open source * Security is super crispy * OS is flexible * No cost * Does have server-based deployment method for host   Disadvantages:   * High learning curve * Poor support for games | Advantages:   * Most common operating system. * More support for games, and * Staff has the most familiarity with this OS. * Does have server-based deployment method for host   Disadvantages:   * Not open source * Will cost for our client to host | Advantage:   * Portability * Does have server-based deployment method for host * For iPhone same advantage as mac   Disadvantage:   * Poor security * Program working is dependent on cell service most of the time |
| **Client Side** | * Decent expertise is needed (mainly familiarity is Mac OS) * Cost average, we just bought all those Macs last week. * As I suggested earlier let’s create a smaller team of people who mainly use Macs and have them write our main code on Swift. | * High Expertise required * Engineers who are familiar will lead Linux side * Cost negligible, Linux is open source | * Expertise on Windows is minimal most common OS. * Cost is average, we mainly use Windows * Must be delivered as a modern, responsive HTML interface running inside the web browser for desktop clients (Linux, Mac, Windows), as well as on mobile platforms. Each will be capable of communicating with the back-end web application running on the server. (This goes for every OS) | * Expertise varies depending on mobile device. * Mac team will handle Apple products * Main team will handle Android. |
| **Development Tools** | * Swift is the most relevant programming language when it comes to Apple. * Licensing costs not an issue Swift is open source * Development tool will be Xcode. * Create a smaller team of people who use Macs and have them write our main code with Swift. | * Default IDE is eclipse * Libraries are different * We will not use the default language * Will use HTML/CSS/JS with VSC. (we’re already paying for it). * Side team not needed but experienced engineers please help | * Default IDE is VS and VSC * Suggest VSC, we can use it across all platforms. * It will cost us but we’re already paying for it * Not using default language (Visual Basic) * Download libraries from open source languages C++/Java/Python * Whole team knows windows, the main team will be on this | * Default IDE for Android is Android Studio, we will use * Default Language is Java, we will also use * Default IDE for Mac IOS is Xcode * Default Language is Swift * Both Open Source no cost |

## 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 like our starting operating platform to be Windows. It’s the one I am most familiar with and I believe most of our team feels this way also. If we start with Windows we can expand to Linux and mobile devices easily. Windows is the most cost-efficient and the OS where we can get our project running immediately.
2. **Operating Systems Architectures**: We are going to use all the Windows applications and libraries to our advantage. We don’t have to do anything fancy with the user-interface just use what Windows gives us. Windows applications have a variety of uses like communication and security.
3. **Storage Management**: We can use the built-in Windows storage sense system. This is on by default and comes with Windows 10 and 11. Storage sense will clean up any unnecessary files when it runs. Storage sense makes it very user-friendly to manage files on a machine/hard drive. We can still use a cloud system for our storage but since the game isn’t very heavy that might not be needed.
4. **Memory Management**: We will use lower resolution pictures for better memory management. I was thinking around 720p, the game can run at a simple 30 fps since we aren’t making anything triple AAA. Windows uses simple memory allocation in which programs and services are assigned with physical or virtual memory space. We can create a folder that serves as directory for all our pictures. For now we’ll keep the pictures at a 720p as I mentioned before but we can later update to make the quality a higher resolution.
5. **Distributed Systems and Networks:** Our client wants to make their game cross-platform. This combined with the fact that we are doing 2D game that isn’t all around complicated, I am going to put forward the idea of using the game engine Unity for our needs. Unity supports all the platforms we are attempting to be a part of and lets them communicate with each other. Konami did a sensational job using Unity to create *Yu-Gi-Oh: Master Duel* and I believe we can recreate that success here. Master Duel is out for PC, PlayStation 4, PlayStation 5, Xbox Series S, Xbox Series X, Android and IOS. All the platforms can play with one another, users can log into any device with their user info and they’ll have access to their own personal account. We also need to focus on having dedicated servers so our connection can be strong on all platforms.
6. **Security**: Windows does come with built-in protection and to keep the project as cost-effective as possible I’m going to recommend, we use it. Initially I was against the thought of using the built-in security due to the fact that Windows is a common OS so malware was already invented to bypass its security features, but Microsoft keeps pushing updates to continuously fortify its security. This made the built-in security a reliable feature.