

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## Table of Contents

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

[**Table of Contents 2**](#_30j0zll)

[**Document Revision History 2**](#_3znysh7)

[**Executive Summary 3**](#_2et92p0)

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

[**Design Constraints 3**](#_1t3h5sf)

[**System Architecture View 3**](#_4d34og8)

[**Domain Model 3**](#_2s8eyo1)

[**Evaluation 4**](#_17dp8vu)

[**Recommendations 5**](#_26in1rg)

## [Document Revision History](#_lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 5/15/2023 | Dakota Wolfe | Added summary, requirements, design constraints, system architecture, domain model, evaluation, recommendations. |
| 1.1 | 6/3/2023 | Dakota Wolfe | Added to evaluation, removed ‘<>’ characters. |
| 1.2 | 6/13/2023 | Dakota Wolfe | Added recommendations page. |

## [Executive Summary](#_35nkun2)

The Gaming Room, a video game development company, wishes to expand their web-based game, Draw It or Lose It, to other platforms (it is currently only on Android). This will certainly bring a larger audience, and more revenue. However, there are some requirements that must be met in order for this project to be a success. The game must have one or more teams, with multiple players on each team, the team names must be unique, and only one instance of the game can exist in memory at any given time.

## Requirements

* (Technical) A game will have the ability to have one or more teams involved.
* (Technical) Each team will have multiple players assigned to it.
* (Technical) Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
* (Technical) Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.

## [Design Constraints](#_1ksv4uv)

The primary constraint of this project is getting the game to meet all of the requirements above, on all platforms. The game runs on Android currently, so it must be ported to the other major mobile operating system, IOS. The game must also be compatible with the three major PC operating systems, Windows, MacOS, and Linux.

## [System Architecture View](#_44sinio)

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](#_2jxsxqh)

The classes named Game, Team, and Player, inherit information from the superclass, Entity, which shares the values name, and id. There are also examples of classes which reference other classes. This is evident in Team’s reference of Player, Game’s reference of Team, and GameService’s reference of Game.

**"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](#_z337ya)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac is easy to use, has plenty of support, and is a stable platform. However, it is expensive, and does not have as large of an audience as Windows. Mac offers server based deployment through tools such as macOS Server, and Apache web server. | Linux is the cheapest platform, as it is open-source, and is a stable platform, similar to MacOS. However, Linux can be more difficult to use. Linux offers server based deployment through tools such as CentOS and Ubuntu Server. | Windows is easy to use, reaches the largest audience, and has plenty of support. However, it is expensive, and has a history of bugs and crashes. Windows offers server based deployment through Windows Server. | Mobile Devices would be easily compatible, cheap, and reach a large audience. However, specifications are the worst, server position is inconsistent, and there is poor security. There are no native options for server deployment on Android or IOS. |
| **Client Side** | Mac would be the most expensive (along with Windows) to implement. Mac would be slightly more difficult to implement than Windows, but much easier than Linux. Ensure compatibility with all web browsers (extra emphasis should be placed on compatibility with Safari) | Linux would be the cheapest to implement. However, it is easily the most difficult to work with. Thus, it would take the most time to implement. Use frameworks with cross-browser compatibility. | Windows would be the most expensive (along with Mac) to implement. Windows would be one of the easiest to work with. Thus, time to implement would be minimal. Ensure compatibility with all web browsers (extra emphasis on Microsoft Edge and Internet Explorer). | Cost should be relatively low, as the game already exists on Android. Difficulty and time to implement should be similar to that of MacOS. Use React Native to ensure browser compatibility (with emphasis on Chrome, Safari, and Samsung Internet). |
| **Development Tools** | Swift is the primary programming language to use when using Apple devices. This can be used with several IDEs, such as Xcode. There are no costs associated with the tools mentioned. The development team must be comfortable working with these tools, as they are not used on Linux or Windows. | Linux is most often associated with C or C++. IDEs that work well with Linux and C / C++ are QtCreator and KDevelop. There are no costs associated with the tools mentioned. The development team must be comfortable working with these tools, as they are not commonly used on Mac. While C++ is common on Windows, the IDEs are not. | The best programming language for Windows is C++. Visual Studio is the go to IDE for this language and operating system. There are no costs associated with these tools, unless the professional or enterprise editions of VS code are used, then there is a fee. The development team should be comfortable with Visual Studio, as it is on every other PC OS. Comfort with C++ is also a necessity. | Kotlin is the go to language for Android app development, and Swift is the go to for IOS. Android studio is the IDE for Kotlin, and Xcode is the IDE for IOS. Android is open-source. IOS is free but locked to Apple devices. Separate development teams may be necessary if one is not familiar with the tools associated with IOS and Android. |

## Recommendations

1. **Operating Platform**: I would recommend Windows, as it has the largest user base, and is very easy to use / understand. There are also plenty of options in regards to moving the application from Windows to other platforms. Such as UWP, WSL and .Net.
2. **Operating Systems Architectures**: Windows uses applications with a graphical user interface. Windows contains Windows Kernel which provides services such as memory management, and device drivers. One Core is another part of Windows which unifies all devices on the Windows platform, making it easier to develop for all different types of Windows devices. UWP is a tool that allows developers to create projects for several device types. WSL is a tool used to run a Linux environment on Windows.
3. **Storage Management**: Windows has a robust storage management system. Windows uses an app named “file explorer” which gives a detailed layout of file locations, as well as storage space available and being used. Storage sense is another Windows feature which helps optimize storage space by removing unnecessary files.
4. **Memory Management**: Windows has a few different tools which help with memory management. One of those tools is referred to as virtual memory. This allows the system to use additional hard drive space to extend the amount of available RAM. Memory manager is another significant tool used in Windows to manage the available memory.
5. **Distributed Systems and Networks**: .Net is an open-source cross-platform tool by Microsoft used for the development of games. This would accomplish the goal of creating a game accessible for all platforms. If player count gets too high, the company will need to ensure their servers are strong enough to keep up.
6. **Security**: Windows uses a built-in firewall and antivirus. Windows Defender Firewall is used to monitor network traffic, and prevent anything malicious from getting in. While Windows Defender Antivirus takes advantage of file scanning to ensure the system is free of malicious software.