

<Name-of-Software-Application>

# **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 | 06/05/2023 | Jared Gonsalves | Initial Version |

**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)

<Write a summary to introduce the software design problem and present a solution. Be sure to provide the client with any critical information they must know to proceed with the process you are proposing.>

The Gaming Room seeks to develop a multiplatform game titled “Draw it or Lose it”, a team-based puzzle game designed in reference to the 1980s television show “Win, Lose, or Draw”. They currently do not understand how to set up the environment for the game. We are going to propose a web-hosted solution and design it.

## Requirements

*<* Please note: While this section is not being assessed, it will support your outline of the design constraints below. *In your summary, identify each of the client’s business and technical requirements in a clear and concise manner.>*

(The following is directly from the assignment summary)

* A game will have the ability to have one or more teams involved.
* Each team will have multiple players assigned to it.
* Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
* 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](#_2et92p0)

<Identify the design constraints for developing the game application in a web-based distributed environment and explain the implications of the design constraints on application development.>

The game will need to be able to be supported on a variety of devices, so the language(s) it’s written must be compatible. It needs to have a good UI that’s easy to follow and works across all device types. It needs to have good scalability to account for both UX and future update implementation. The short of this is that it must be functional, pretty, and easy to use.

## [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)

<Describe the UML class diagram provided below. Explain how the classes relate to each other. Identify any object-oriented programming principles that are demonstrated in the diagram and how they are used to fulfill the software requirements efficiently.>

Present in this UML diagram are seven classes: Entity, Player, Team, Game, GameService, ProgramDriver, and SingletonTester. Entity, a superclass, inherits all the attributes of its subclasses, Game, Team, and Player. The relationship between the bottom four classes in the chart are 0 to many relationships, meaning that you can have none or as many as you’d like present. ProgramDriver exists within the program to execute it and SingletonTester calls a simple test function upon execution. Team, Player, and Game all create unique identifiers relating to their singleton instance. GameService fetches all this information and returns it to the program through get functions.

**"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** | MacOS is bound to Apple devices and Apple devices only. Apple has airtight security, an easy-to-use GUI, and great tools for development and webhosting. The problem comes with cost as Mac devices are expensive and you would ideally want a higher tiered computer for this. | Linux, as an operating system, is its own different beast. It’s cost effective due to it being open source, but there is a large learning curve to handling it. I do not see an issue with Linux being able to perform what the client needs from it. | Windows applications have been developed for web-use for decades now. It supports most languages and IDEs and has a very trusted line of operating systems. It’s robust and customizable, allowing total control over who touches what in the system. Windows newest versions are notoriously vulnerable at first, recommended to use the previous version until fixes occur. | At the end of the day, this is a phone running a server. Cellphones aren’t exactly the most secure devices on the market and are vulnerable to attack, especially if proper measures aren’t taken pre-deployment. The upside is that you can take it wherever you need to and launch it whenever you need to, but seeing as it’s a game, you wouldn’t want to do this. |
| **Client Side** | Developing on Mac is just like developing on Windows but with less available tools. Over time, many tools and languages that were exclusive to Mac have gone to Windows and Linux, like Swift. Ideally, we’d write this in Java or Swift so that it’s compatible with the big three OS, on top of it being compatible with Android and IOS. | You’d have to learn how to navigate Linux and utilize its terminal prompts to the fullest to develop for it. In my experience, people generally do not touch Linux machines due to compatibility issues with most software. Virtual machines are often run to utilize software available to the other operating systems, primarily Windows. It is free, so if you spend the time learning on top of the time spent developing, you’ll have a low maintenance project. | Windows as a client to work on is just good. Most people who develop software are extremely familiar with windows clients. It's host to innumerable IDEs and is compatible with most programming languages. It’s mostly affordable to just stick with Windows. | Mobile apps are cheap to make and can be managed very easily using the right tools. IOS and Android are compatible with many programming languages that are used by the big three. |
| **Development Tools** | IDEs:  NetBeans  Xcode  VS Code  Terminal  Notepad++  Languages:  Objective-C  Java  Python  Ruby  Swift | IDEs:  VS Code  Sublime  Terminal  Notepad++  Most others…  Languages:  All | IDEs:  Visual Studio  Xcode  NetBeans  VS Code  Notepad++  Most others…  Languages: Objective-C  Java  Python  Ruby  Swift  C++  Most others… | Utilizing tools like Android Studio, Swift, Xcode, or React, we can easily develop mobile devices. Amazon Web Services also offers a hosted option, which given the price point seems ideal. |

## 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**: <Recommend an appropriate operating platform that will allow The Gaming Room to expand Draw It or Lose It to other computing environments.>
   1. Windows would be the preferred environment for our project. It’s cheap, easy to use, and has all the necessary tools we need for the job. Utilizing this OS, we can design, develop, and maintain our software all in one place as well as distribute it.
2. **Operating Systems Architectures**: <Describe the details of the chosen operating platform architectures.>
   1. Utilizing the Windows API, every application developed on a Windows machine has access to necessary and vital resources. A thread handles the instructions within a process and executes it as it is read. These threads contain all sorts of content necessary to execute that it is given. Processes are the executable files and resources required to run the program.
3. **Storage Management**: <Identify an appropriate storage management system to be used with the recommended operating platform.>
   1. I’d recommend Microsoft Azure Storage to handle our data. Microsoft handles all the stuff we don’t want to for a fee, and we get to enjoy the benefits of cloud-based storage. If this isn’t acceptable, we could utilize a very large SSD to ensure we have enough room for the images the game needs as well as our code and room to handle everything else.
4. **Memory Management**: <Explain how the recommended operating platform uses memory management techniques for the Draw It or Lose It software.>
   1. Windows has a very robust memory management system built into it. Some of the tools Windows uses includes, but is not limited to, paging, utilizing the heap, file mapping, memory reservation and reallocation, and much more.
5. **Distributed Systems and Networks**: <Knowing that the client would like Draw It or Lose It to communicate between various platforms, explain how this may be accomplished with distributed software and the network that connects the devices. Consider the dependencies between the components within the distributed systems and networks (connectivity, outages, and so on).>
   1. To start, every device would need to be connected to the internet, would have to have power, and able to receive data being distributed to it from our server. If a player disconnects, the game most likely will as well seeing as how it’s a team game. That, or the player would have to reconnect to their singleton instance if their unique identifiers don’t vanish on disconnect. We can distribute the app through open-source places like Google Play, the iOS store, or even paid options like Steam, the Epic games store, or the Xbox/PlayStation marketplaces, etc. Using the proper internet protocols to ensure strong connections between devices will be important and testing for that can be done.
6. **Security**: <Security is a must-have for the client. Explain how to protect user information on and between various platforms. Consider the user protection and security capabilities of the recommended operating platform.>
   1. Windows has baked in security tools like Windows Firewall, ability to install anti-virus, and multiple ways to protect your device like adding a PIN, Windows Hello, and having a complex enough password. Two-factor authentication is also possible through the Windows Authenticator mobile app as another way to secure user data, and 2FA can be an option that users could opt into later.