

Draw it or Lose it

# **CS 230 Project Software Design Template Jack Coster**

Version 1.2

## 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 | 11/12/23 | Jack Coster | Created the Executive Summary and Design Constraints |
| 1.1 | 11/25/23 | Jack Coster | Comparison of different operating systems |
| 1.2 | 12/10/23 | Jack Coster | Developed design 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 team at The Gaming Room has asked us to develop the web version of their game “Draw it or Lose it” which is loosely based off the 1980’s tv game “Win, Lose, Draw”. The game involves teams guessing an image being rendered from a stock image library over the course of one-minute rounds. The drawings are fully rendered at the 30 second market and if the guessing team cannot complete the puzzle, the opposing team will have 15 seconds to guess and steal the points. The Gaming Room team does not know how to set up the environment and has tasked us with streamlining development.

## Requirements

* The game must be web based and accessible across all modern operating systems: Windows, IOS, Android, etc.
* The game must allow for online connection between players on different devices for multiplayer play.
* The game must be able to store in game data to track scores during a session.
* The game must let multiple players connect to the same instance at once,

## [Design Constraints](#_2et92p0)

* Different operating systems may require unique code blocks to present a consistent user interphase across different devices. For example, the mobile view may require responsive design to render correctly on smaller screens than the windows version.
* Users’ internet connection will have to be adequate to connect to game servers without having significant input lag. This will be further impacted if players are competing from different areas and connecting to different servers.
* If a player loses connection to the internet during play, in progress results may reset or the session may end.
* The game will need to have scalable in session storage that can get expensive to store player scores and answers.

## [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)

In this model, the Entity class is the base class for all other objects in the game. This holds the id and name properties which the “Game”, “Team”, and “Player” classes will all inherit. The “GameSevice” class is the system manager in this game used to handle the individual game instances and this game can have 0 too many instances. The “Game” and “Team” classes have a similar relationship as well as “Team” to “Player” with 0 too many instances. The “ProgramDriver” class initializes the only instance of “GameService” and uses the “SingletonTester” class to perform the necessary validation during run-time.

**"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** | Characteristics: Powerful hardware, stable OS with constant updates, solid developer support.  Advantages: Reliable, secure, Great UI experience.  Weakness: Expensive, lots of proprietary software and licensing specific to Mac. | Characteristics/ Advantages: Very customizable, extensive developer community, great hardware support, open-source language, scalable and cost effective. Disadvantages: Less developer support from Linux, harder to learn as a beginner. | Most common OS and largest user base. Extensive hardware support and low cost to develop. Potential for issues with updates that come infrequently and not as cutting edge as Mac OSX due to wide range of userbase worldwide. Most likely to have security issues. | Device specific languages for android or IOS require specific code setup if you want the app hosted on the device app store it must meet certain standards and be approved for launch. Building for less powerful hardware requires code to be optimized to run on mobile devices. |
| **Client Side** | Knowledge of Mac/ Apple specific UI and system configurations needed for development. As mentioned above, entry cost on even basic Macbook Air way more than PC. Once development environment is set up, deployment and development very smooth process. Mac development may be tailored towards Safari for browser support, but Mac supports deployment to all platforms. | Lower cost to develop, but may require brining in additional short term developers as a lot of linux frameworks do not have extensive software support outside the dev community. Installing a Linux environment on your machine also requires additional setup as Linux machines are sold in store, hardrive on PC or Mac must be portioned with a Linux boot drive so HD space can be smaller. | Lowest cost and expertise needed to develop on Windows. Tons of resources for learning and developer support. Easy set up for environment and fairly seamless connection to your development server through Git or other tools. Windows developed programs should run on all devices without issue besides mobile devices if they are mobile native. | Mobile specific UI requires client side touch screen code. Must also be cognizant of battery life constraints for the user on client-side mobile code. Can use Mac to develop for IOS devices and Windows to build for Android devices so development cost will vary based on device. Android and IOS may require separate development cycles as well as platforms are not interchangeable. |
| **Development Tools** | Swift is the internal coding language developed for Mac OSX. Supports development of major coding languages like Java, Python, Ruby, C#, Javascript etc. Xcode is the go to IDE for Mac users developed internally by apple. VS Code, JetBrains and Eclipse other popular IDE for Mac users. X Code available for free in app store. | Again Eclipse, VS Code, JetBrains IDEs like PyCharm. Linux also provides lots of package managers and command line dev tools. Mostly known for C, C++, Python and Java development. Being open source, no cost to develop on Linux besides potentially setting up VM environments to deploy in. | Biggest IDE for Windows is Visual Studio, the more powerful project manger compared to VS Code and other options are plentiful. Every main programming language supported by Windows. Visual Studio requires licensing cost. | Apple devices will require Xcode to develop swift UI and UI kit for IOS. Windows version available as well. Android studio needed for Android development can be used on Windows or Linux. May have cost to host on App store for IOS and Android. |

## 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**:

When deciding what to deploy the game on, we must consider which platform will reach the largest possible player base and have the easiest development process. Because of this I would recommend building the game on PC and if possible, make it web based using a combination of HTML/ CSS and JavaScript libraries. This architecture provides a flexible option that can run on most web browsers across all devices including IOS and Android. This decision also gives the development team a large developer community to interact with for help during the design phase and web-based games are usually easier to build with a shorter development time. Browser specific considerations may have to be made along with some responsive design for mobile users, but these can be implemented during the later phases of development.

1. **Operating Systems Architectures**: Building the game with a variety of JavaScript libraries gives the game a lot of options when considering the needs of a game like this. This game will have a combination of server-side events and client-side events, so making sure the code isn’t too resource intensive to slow down the browser displaying the game is a key detail to remember when building the game. Building the game so that the browser holds temporary memory is another way to make sure the game runs smoothly.
2. **Storage Management**: For a web-based game like this it is most important to consider a storage environment that is secure, available, and scalable. This leads us to using a cloud-based solution like AWS or Azure. Making the game cloud based means we don’t have to worry about allocating additional storage space in a physical data center during peak traffic times. This also is a more cost-effective architecture as cloud-based storage is usually cheaper, especially for a low power application like this. The other main benefit to cloud storage is automatically applying new OS and security patches, and automatic managed backup plans that can re-deploy the game to a new cloud instance in the case of an outage or bug.
3. **Memory Management**: As mentioned above, remembering that running a game on a web browser will be a large strain on the device memory especially if using Chrome. Browser games do present a memory drag on most CPUs, so building the game with the necessary memory assets like load balancing, auto-scaling groups and fault tolerance will help the game run as intended. This is another benefit to hosting the game on a cloud platform as these features are usually built into an AWS or Azure deployment and can be auto adjusted to increase power as more users play the game.
4. **Distributed Systems and Networks**: Having “Draw it or Lose it” communicate between multiple platforms and networks is another feature of having the game hosted on a cloud network. AWS uses a content delivery network (CDN) to reduce load times, reduce costs, increase content availability, and improve security. Players will judge the system based on how fast it loads and how few bugs/ crashes they experience when playing the game. Having the system managed by the CDN keeps these assets managed inside one place and monitoring the health of the game also becomes easier when fully handled within the cloud platform’s CDN.
5. **Security**: The game will be designed to hold small amounts of user data like an email and password to log into their player account before playing against another user. This means we must design the system to have multi factor authentication and strong password requirements. Since it is browser based, the site itself must also use HTTPS authentication and have an up-to-date SSL certificate to make sure the game is always being hosted on the most secure type of website. Finally, inside the cloud platform it is stored on, the CDN can serve regular security updates and any vulnerability patches as needed.