

Draw it or Lose it

# **CS 230 Project Software Design**

Version 3.0

## Table of Contents

[**CS 230 Project Software Design** 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 7**

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 03/19/2024 | Mohammed Khan | Early-stage prototype software development. |
| 2.0 | 4/6/2024 | Mohammed Khan | Defining development requirements |
| 3.0 | 4/16/2024 | Mohammed Khan | Recommendation for system architecture |

**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)

We need to make a game that people can play online. This game should be fun and work well on computers and phones. Our plan is to build a game that can handle lots of players at once and keep their information safe. We'll make sure it runs smoothly even when many people are playing at the same time.

## Requirements

The game needs to be enjoyable and keep players interested. It should let many people play together and allow them to interact. Technically, it must work on web browsers, be secure, and handle lots of players without crashing.

## [Design Constraints](#_2et92p0)

Building the game online means we have to consider different devices and internet connections. We also need to make sure people's information stays private and that the game runs smoothly even with many players online.

## [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)

The UML class diagram provided represents the domain model of the game application. It includes classes such as Game, Player, Team, Entity, and GameService. These classes reflect the entities and relationships within the game domain. Object-oriented programming principles demonstrated include inheritance, encapsulation, and singleton pattern. These principles are utilized to efficiently organize and manage game-related data and functionalities, fulfilling the software requirements effectively.**"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** | Mac operating systems provide a user-friendly interface for developers and integrate well with Apple ecosystem tools. However, they have limited server deployment options, primarily relying on macOS Server, which may require additional licensing costs. | Linux is an open-source operating system known for its stability, security, and flexibility. It offers robust server deployment options, such as Ubuntu Server, making it ideal for hosting web-based software applications. While it is free to use and has minimal licensing costs, Linux requires technical expertise for setup and maintenance. | Windows operating systems are widely used in enterprises and offer seamless integration with Microsoft products. Windows Server provides strong server hosting capabilities but may require significant licensing costs, particularly for large deployments. | Smartphones and tablets are popular for web apps. When deploying on mobile, consider screen sizes, operating systems, and performance. |
| **Client Side** | Development for Mac requires expertise in building responsive HTML interfaces for Safari. Testing across various screen sizes may increase development time slightly, and costs may be higher due to specialized Mac development tools. | Supporting Linux involves ensuring compatibility with web browsers like Firefox, Chrome, and Chromium. Development time and costs may be lower due to open-source tools and community support. | For Windows, compatibility with browsers like Chrome, Firefox, and Edge is essential. Development time and costs are comparable to Mac due to testing across multiple browsers and Windows versions. | iOS and Android development requires expertise, along with a responsive HTML interface for mobile web browsers. Development time and costs are higher due to separate codebases and the complexity of mobile app development. |
| **Development Tools** | For Mac, developers use a program called Xcode. It helps them write code in languages like Swift, Objective-C, and C/C++. They can also use Visual Studio Code, Atom, or Sublime Text. | Linux developers have a lot of choices. They can use Visual Studio Code, Atom, Sublime Text, or other tools. They write code in languages like C/C++, Python, Ruby, PHP, and JavaScript. | Windows developers often use Visual Studio, Visual Studio Code, or JetBrains IDEs. They write code in languages like C/C++, C#, JavaScript, and .NET. | Making apps for mobile devices is different. For iPhones and iPads, developers use Xcode and write code in Swift or Objective-C. For Android devices, they use Android Studio with Java or Kotlin. There are also tools like Xamarin, React Native, and Flutter for making apps that work on both 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**: The Gaming Room should choose a cloud-based system like Amazon Web Services (AWS) or Microsoft Azure. These systems let Draw It or Lose It grow and work on different devices because they can expand and reach people all around the world.
2. **Operating Systems Architectures**: Cloud systems, like AWS or Azure, are made up of many data centers where they keep information. These data centers are in different parts of the world and are connected by fast networks. Inside each data center, there are groups of computers running special programs, like Windows or Linux. These computers can change how much they do based on how many people are using them.
3. **Storage Management**: A cloud storage system like Amazon S3 or Azure Blob Storage would work well. These systems are very safe and can hold a lot of information, like the drawings and player details in Draw It or Lose It, in a place on the internet where everyone can get to them.
4. **Memory Management**: In the cloud, the system manages how much memory each part of the game needs. It does this by giving more or less memory to each part depending on how much work it has to do. This way, everything works as fast as possible without using too much memory.
5. **Distributed Systems and Networks**: To make Draw It or Lose It work on different devices, it can utilize APIs from the cloud system to help different parts of the game talk to each other. Also, things like RESTful APIs or messaging queues can help different parts of the game talk to each other too. It's important to make sure that if something goes wrong with one part of the game, the whole game doesn't stop working. That's why it's important to have backups, balance the load, and be ready to fix problems quickly. This helps the game keep running smoothly even if there are problems with the internet or the devices people are using.
6. **Security**: Cloud systems have good ways to keep information safe. They make sure that no one can read the information while it's moving around the internet, and they keep it safe when it's stored somewhere. They also make sure that only the right people can get to the information. Plus, they watch for any problems and fix them quickly. This way, players can feel safe knowing that their information is protected.