

Draw it or Lose it

# **CS 230 Project Software Design**

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 |
| --- | --- | --- | --- |
| 1.3 | 08/18/24 | Dominoe LaMattina | Draft for software design needs. |

## [Executive Summary](#_sbfa50wo7nsh)

The web-based game Draw it of Lose it was inspired by 1980’s show “Win, Lose, or Draw”. One or more teams will compete to guess what is being drawn from a library of images. There will be four rounds that last one minute, with drawings being fully rendered by 30 minutes. If the team that is up is not able to guess the puzzle, then the other team has 15 seconds to make a guess.

We will review the outline and design constraints for this game to meet client needs, such as the ability to manage multiple teams and players, force unique names, and maintain one instance of the game.

## Requirements

* Support one or more teams with multiple players.
* Team names must be unique.
* Game instance must be unique.
* Each game has four rounds that last one minute.
* Drawing is fully rendered at 30 seconds.
* If the team is unable to guess the puzzle, then the next team is allotted 15 seconds to guess.

## [Design Constraints](#_2et92p0)

* Must be able to manage unique identifiers.
  + Games
  + Teams
  + Players
* Only one instance can exist in memory.
* A strong database system is needed to store data.
* The game needs to be web-based.

## [System Architecture View](#_ilbxbyevv6b6)

* Frontend
  + Web-based
  + HTML, CSS, and JavaScript
* Backend
  + Server handling data
* Database
  + Stores the game information.

## [Domain Model](#_8h2ehzxfam4o)

* Entity
  + Attribute
    - id
    - name
  + Methods
    - Entity()
    - getId()
    - getName()
    - toString()

Entity class is the base class of the program.

* Game
  + Attributes
    - teams
  + Methods
    - Game()
    - addTeam()
    - toString()

Game class stores the list of teams and pulls from Entity.

* Team
  + Attributes
    - players
  + Methods
    - Team()
    - addPlayer()
    - toString()

Team is the list of players that comes from Entity.

* Player
  + Methods
    - Player()
    - toString()

Player is the actual player playing. Which is pulled from Entity.

* GameService
  + Attributes
    - games
    - nextGameId
    - nextPlayerId
    - nextTeamId
    - service
  + Methods
    - GameService()
    - getInstance()
    - addGame()
    - getGame()
    - getGameCount()
    - getNextPlayerId()
    - getNextTeamId()

GameSevice is a singleton that manages instances.

* ProgramDriver
  + Methods
    - main()

ProgramDriver has the main method to rum the application.

* SingletonTester
  + Methods
    - testSingleton()

This SingletonTester, tests the singleton in GameService.

**"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 OS has reliable performance and security.   Advantages:  Stable, built in dev tools such as Xcode.  Weakness:  High cost of hardware, not commonly utilized for a large-scaled server.  Deployment:  Not ideal for web hosting.  Licensing costs:  High with software and hardware. | Customizable, modifiable, and exceptional for web servers.  Advantages:  Stable, trusted security, and open-sourced.  Weakness:  Harder learning curve.  Deployment:  Great for server deployment utilizing Apache, Nginx, etc.  Licensing cost:  Low or close to none since it’s open source. | Windows is easy to use and can be done with enterprise apps.  Advantages:  User friendly and great technical support.  Weakness:  Potentially not as secure, depending on configuration methods.  Deployment:  Strong server deployment support.  Licensing cost:  High costs due to licensing fees. | Cloud services are great for mobile devices since they are scalable.  Advantages:  Easy accessibility and scalability.  Weakness:  Resources are limited on mobile devices.  Deployment:  Cloud services like AWS and Azure.  Licensing cost:  They vary by provider. |
| **Client Side** | Mac OS has great graphics and performs very well.  Advantages:  Top notch for rich interfaces and media content.  Weakness:  High costs for equipment and requires Safari optimization.  Development:  HTML5, CSS3, JavaScript for browsers. | Linux is cost efficient and flexible.  Advantages:  Low software costs. Compatibility with various distributions.  Weakness:  Need to knowledgeable in Linux.  Development:  Compatible with browsers Chrome and Firefox. | Most users and developers are familiar with Windows.  Advantages:  Lots of options for tools. Lot of developers are familiar with Windows.  Weaknesses:  Licensing cost and potential compatibility issues with old versions.  Development:  Must be tested on IE, Edge, Firefox, and Chrome. | Mobile development is heavily involved in responsive design and performance.  Advantages:  Able to reach a larger audience base.  Weakness:  Android and iOS have different development requirements.  Development:  Need to test across devices. |
| **Development Tools** | Programming languages and tools: Swift programming, JavaScript programming, visual studios, terminal use.  Development impact:  High end tools that are dedicated Mac OS specific development.  Licensing:  Some tools are free, others like IDE have costs. | Programming languages and tools:  Java, python, JavaScript, C++, Eclipse, visual studio.  Development impact: Open-source tools which lower costs. Versatile skills are needed.  Licensing:  Minimal costs since most tools are open-source. | Programming languages and tools:  C#, .NET, JavaScript, Python, Visual studios, PyCharm, PowerShell  Development impact:  Might need to involve larger teams for Windows development.  Licensing:  Visual Studios and Windows licensing is expensive. | Programming languages and tools:  Swift for iOS, Kotlin/Java, Xcode, Android studio  Development impact:  Requires extensive knowledge in mobile device platforms.  Licensing:  Xcode is free, Android Studio is free, but with the APP store there are fees. |

## 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**: With a web-based platform will be the most suitable choice. The game will be accessible from all operating systems and mobile devices. The focus would be to ensure they are functional on all web browsing clients. This platform leverages the universality of web browsers, which makes it easier to update and maintain the game.
2. **Operating Systems Architectures**: The architecture is a web-based system, which is accessed via web-browsers like Google Chrome, Safari, Firefox, and Microsoft Edge. Making the game web-based allows the game to function independently and focuses on ensuring the compatibility with web technologies like HTML, CSS, and JavaScript.
3. **Storage Management**: Cloud-based storage solution like Amazon S3 or Google cloud services is recommended. These services provide scalable storage that will grow along with the game’s needs. Cloud storage also integrates with web applications, which allows seamless management of the games data, user information, and other critical assets.
4. **Memory Management**: With a web-based environment, memory management can be optimized by implementing caching mechanisms. This will reduce database load and improve performance by storing frequently accessed data in memory, which will allow for faster retrieval times. Cloud platforms often offer built-in memory management tools that will be leveraged to ensure the game runs smooth even as the game’s user demands scales.
5. **Distributed Systems and Networks**: We will need to manage API requests between microservices and clients. Implement load balancing to distribute traffic. Since there is a requirement to communicate between various platforms, we will need to utilize microservices. These services will handle specific functions within the game, communicating via APIs. Load balancing will be implemented to help distribute traffic evenly across servers. The user of cloud-based services like AWS or Google cloud can help simplify the management of these distributed systems.
6. **Security**: Security is critical for protecting user data across platforms. The game will utilize HTTPS to encrypt data transmission, so user information remains secure. We will use methods like OAuth2 to verify game player identities. There will also be regular security audits, along with real-time monitoring, to identify and address potential vulnerabilities.