

<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 | 09/16/2024 | Mustafa Alsouqi | Filled in all responses. |
| 2.0 | 09/28/2024 | Mustafa Alsouqi | Revised responses for project two. |
| 3.0 | 10/18/2024 | Mustafa Alsouqi | Revised responses for project Three. |

**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 Gaming Room has asked us to build a web-based game called Draw It or Lose It. The game is an old TV show where teams guess what's being drawn but instead of players drawing, we'll use a library of pre-made images. The game will need to handle multiple teams, each with several players, and make sure every team and game name is unique. Also, we need to ensure that only one game can run at a time, using a singleton pattern to make this happen.

## Requirements

* Unique Names: Every game and team need a unique name.
* Multiple Teams and Players: The game needs to support several teams, with each team having multiple players.
* One Game Instance: Only one game can be in memory at a time.
* Web-Based: The game will be online, so it must work well in a web environment.

## [Design Constraints](#_2et92p0)

* Scalability: The game might need to handle a lot of teams and players, so we need to use efficient lists and unique IDs for each game, team, and player.
* Singleton Pattern: To ensure only one game is running at a time, we’ll a singleton pattern. This is a way to make sure only one instance of the game’s main service is created.
* Unique Names: We need to make sure no two teams or games have the same name, which means checking names before creating a new game or team.
* Multi-user Environment: Since this is an online game, multiple people will be interacting with it at the same time. We’ll need to manage sessions and make sure the game stays consistent for everyone.
* Performance: The game will render images quickly, so we need fast and efficient code to keep everything running smoothly.

## [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)

* ProgramDriver: This is where the game starts; it contains the main method and works with SingletonTester to check that the game only runs one instance at a time.
* SingletonTester: This class checks that our GameService is correctly following the singleton pattern, ensuring there’s only ever one instance.
* Entity: This is a base class that stores shared attributes like id and name, which are passed down to Game, Team, and Player to avoid repeating code.
* GameService: This is the brain of the game, handling everything like creating games and teams. It also uses the singleton pattern to ensure only one instance is ever created.
* Game: Represents an actual game with multiple teams. It lets us add teams and manage the list of teams for that game.
* Team: Represents a team in the game, which can have multiple players. Teams are part of a game, and this class allows adding players to the team.
* Player: Represents a player on a team. Every player belongs to a team, and this class keeps track of their details.

OOP concepts used:

* Inheritance: Game, Team, and Player all get common attributes from Entity, which keeps our code cleaner and easier to maintain.
* Singleton Pattern: GameService uses this to make sure there’s only one instance of the service, which keeps things consistent across the game.
* Encapsulation: Each class has its own data and methods to keep things organized, so one class doesn’t accidentally mess with another.
* Associations: The lines between the classes show how everything is connected—a game has teams, and teams have players. This reflects how the game works in real life and keeps everything organized.

**"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 is not as widely used for server-side applications due to the higher cost of hardware and limited enterprise level support compared to Linux. | Linux is the most popular platform for web servers, offering strong performance, security, and flexibility. It is open-source and highly customizable, making it a go to for many developers. | Windows has strong support for web hosting and enterprise applications, though it is typically more expensive and less secure compared to Linux. | Mobile devices aren’t used to host web based applications directly |
| **Client Side** | Developing for Mac clients requires attention to the Apple ecosystem. Costs are high because of the specialized hardware and software, but Apple’s strict design standards and consistent hardware ensure a smooth client experience. | Linux offers flexibility on the client side, but it’s not as popular with general users. | Windows is the most common client platform, especially in business environments. Developing for Windows is relatively straightforward. | Mobile devices are increasingly important as clients for web-based apps. Android and IOS are the most widely used operating systems today. |
| **Development Tools** | Mac development uses languages like Swift and Objective-C, particularly for macOS and iOS apps. Xcode is the primary IDE. It also supports web based development through tools Node.js, Java, and Python. | Linex is highly customizable and open-source, but developers need to be comfortable working in the command line. IDEs like Eclipse, PyCharm, and Visual Studio Code are commonly used. | Windows is well suited for enterprise software but can be more resource intensive. Visual Studio is one of the most robust and popular IDEs for Windows development. | Mobile devices arnt used for development tools. Unless for testing purposes. |

## 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**:

I recommend Linux as the best operating platform for expanding Draw It or Lose It. Linux is flexible, widely used for web servers, and works well with other systems like Windows and macOS. It's also free, which makes it a cost-effective choice. By choosing Linux, The Gaming Room can easily run the game across different devices and platforms, ensuring smooth expansion and cross-platform compatibility.

1. **Operating Systems Architectures**:

Linux uses a monolithic kernel architecture, meaning the core part of the system controls everything from memory to input/output operations. This setup makes Linux highly efficient and stable, especially for large-scale applications. It’s also modular, meaning you can tweak and add parts based on what the game needs, improving performance without unnecessary bloat.

1. **Storage Management**:

For storing game data such as user accounts and game info, I suggest using a relational database system such as MySQL. MYSQL is reliable and work well with Linux. MYSQL can handle data quickly and securely, ensuring that the game runs smoothly even as it grows. Both systems also allow for easy scaling, meaning they can keep up with increasing demand as more players join the game.

1. **Memory Management**:

Linux is great at managing memory. It uses paging and swapping to make sure active processes have enough memory, while less active ones are temporarily moved to disk. This helps the game run smoothly even with many users playing at once. Linux can use virtual memory to extend physical memory, making it more efficient for handling lots of data during gameplay.

1. **Distributed Systems and Networks**:

To allow the game to communicate between different platforms, Linux supports a distributed system setup. This means the game can run on multiple servers and still work together as one system. Communication between different parts of the game can happen through APIs, and using cloud services helps ensure the game stays online even if one server goes down. Overall, Linux helps the game connect across different devices and platforms without hiccups.

1. **Security**:

Linux is known for strong security. It offers solid protection through user permissions, encryption, and firewalls, which keep sensitive data safe. For Draw It or Lose It, we can secure communication between the server and users through encryption, so no one can intercept private info like player details. Additionally, Linux supports multi-factor authentication to further protect user accounts and data, making it a reliable and secure choice for the game.