

The Gaming Room

# **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 | 09/11/2023 | Ryan Hubbuck | Outlined the Executive Summary, Design Constraints, and described the Domain Model per the UML diagram. |
| 1.1 | 09/29/2023 | Ryan Hubbuck | Completed evaluation of server-side, client-side, and development tools for Mac, Windows, Linux, and mobile operating systems. |
| 1.2 | 10/10/2023 | Ryan Hubbuck | Provided recommendations on the operating platform, storage management, memory management, security, and distributed network management. |

**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 is seeking to create a web-based game that can serve multiple platforms based on their already existing Android app, Draw It or Lose It. The basic concept of the game is that teams will compete in 4 rounds of 1 minute intervals to guess what is being drawn. Images are fully rendered by the 30 second mark and if a team does not correctly guess the puzzle, the remaining teams have 15 seconds to guess the puzzle. The software created must work dependably and efficiently on all operating systems.

## Requirements

The business requirement is to have a game that works across all operating systems to maximize the player base. The technical requirements are that each team and game must have a unique name, there can only be a single instance of a game at a given time, each game must have one or more teams, and each team must have multiple players.

## [Design Constraints](#_2et92p0)

Each game must have 1 or more teams involved

Each team will have multiple players

Team names must be unique to allow players to check whether team names are in use

Game names must be unique and only 1 instance of the game can exist in memory at a given time

There must be unique identifiers for each instance of a game, team, or player

## [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 Entity class is the base class from which Game, Team, and Player inherit from. This allows Game, Team, and Player to inherit attributes and methods from Entity and extend that functionality. These three child classes are associated with each in a way that the Game and Team relationship is ‘0 to many’ and the Team and Player relationship is also ‘0 to many’. This means that a game may contain multiple teams and a team may contain multiple players. Additionally, a GameService can have multiple games and is a ‘0 to many’ relationship as well. The ProgramDriver class uses SingletonTester to ensure there is only a single instance of the program running. Of the four object-oriented programming principles, abstraction can be seen by classes such as Entity, GameService, Game, and Team containing private attributes that can not be seen or accessed by other parts of the program. Additionally, inheritance is present between Game, Team, and Player and the parent class Entity. Encapsulation is present within each class because the program is broken up into individual classes to keep data separated into individual units. Finally, polymorphism can be observed by Game, Team, and Player using methods inherited from Entity.

**"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** | Less support and compatibility compared to Linux and Windows. User-friendly GUI. High degree of hardware protection/ security. Limited hardware configurations but has built-in Apache commands. Mod-high cost. | Cost efficient hosting but may require a higher degree of expertise due to uniqueness and reliance upon CLI. Requires less resources to run this OS on the server compared to others. Server configuration completed efficiently via CLI. Stable for long periods of time. Lowest cost. | More expensive to run on server-side but has a high degree of support and stability. User-friendly GUI. Virtualization to improve compatibility. Mod-high cost. | Mobile devices themselves are not used for hosting, but systems such as AWebServer (Apache) and various cloud-based platforms are the most common for hosting mobile apps. Cloud hosting is higher costing than other systems. |
| **Client Side** | Moderate level of time and expertise required for development. More expensive than Linux to develop. Increased cost developing on a Mac and within Xcode with Swift. Similar user interface to Windows. | Higher degree of time and expertise required of users compared to other systems. Heavy reliance on CLI. More cost effective than other systems, as it is open-sourced. Higher degree of variability among several Linux distributions, so security and stability will be a concern – shared and dynamic linking can save resource strain. | Minimal level of time and expertise compared to other systems. Higher level of cost when compared to Linux. May need .NET (Visual Studio) expertise. Similar user interface to Mac. High degree of support and can virtualize other machines/ systems. | Higher level of time and expertise required for mobile development. Developers need expertise using languages such as Swift and Kotlin. Highest degree of client-side flexibility for users and variance in screen size and resolutions, which developers need to account for. |
| **Development Tools** | All major programming languages (HTML, CSS, JavaScript, Python, Java) and IDEs such as VSCode, Pycharm, & Eclipse. Swift is a unique language to mac/iOS development. Xcode is macOS’ formal IDE and is $99/year. | All major programming languages (HTML, CSS, JavaScript, Python, Java) and IDEs such as VSCode, Pycharm, & Eclipse. More reliance upon terminal/CLI and is usually the preferred OS and is referenced in the majority of development resources online. Most IDE’s are free. | All major programming languages (HTML, CSS, JavaScript, Python, Java and IDEs such as Visual Studio, VSCode, Pycharm, & Eclipse. Microsoft Visual Studio commonly used for C# and C++ development – integrates well due to both being Microsoft. Most IDE’s are free. Visual Studio charges for large-scale development. | All major languages in addition to Swift for iOS and Kotlin for Android. Xcode is iOS’ formal IDE and is $99/year. Android studio is free. React-native and Flask are also common, but cost-effective. |

## 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 operating platform to allow Draw It or Lose It to expand to other environments. A Linux operating system such as Ubuntu will provide a cost-effective system for our server and is stable and secure due to the amount of third-party support.
2. **Operating Systems Architectures**: The architecture of the Linux system is a layered structure that comprises several components, each with a specific function. These layers are the Kernal, System Library, Hardware layer, and Shell utility. The Kernal is responsible for the major actions of the operating system and will require the hiring of software developers experienced in interacting with the Linux system.
3. **Storage Management**: I recommend direct-attached SSD, and more specifically NVMe storage if we can afford it. SSDs are faster, more compact, and require less energy resources to run. This will provide the best user experience while reducing system stress to allow resources to be allocated to running the game. NVMe is a newer version of SSD drives and is even faster than traditional SSDs.
4. **Memory Management**: A few of the key memory management techniques utilized by Linux is paging, nodes, and virtual memory. Nodes refer to physical or logical groupings of memory identified by a unique address. Memory is divided into discrete chunks and each node is responsible for its own memory allocation. Paging divides accessible system memory into discrete ‘pages’ with unique addresses. Each page is mapped to virtual addresses. Virtual memory allows for secondary memory to be used as a part of primary memory in case all available RAM is being used. These techniques work together to ensure efficient memory usage and reduce the likelihood of memory leaks or crashing.
5. **Distributed Systems and Networks**: Knowing that the client wants Draw It or Lose It to communicate between various platforms, we could implement a distributed system in which a collection of processors do not share memory but instead communicate via high-speed buses and telephone lines. We could implement a fully connected system designed in a star topology in which all nodes are connected to a central hub or router. This design is easier to set up, less expensive, and is easier to maintain. I believe this design can suit the needs of a company of our size and will reduce latency and increase fault tolerance. If a single or few machines encounter an issue the system will still be functional as the other nodes increase workload. To protect against power outages, we should invest in a backup power supply such as a UPS. A UPS can ensure power to the system in case of building wide power outages.
6. **Security**: To ensure client security I recommend utilizing end-to-end data encryption, role-based access control (RBAC), and the implementation of honeypots. With end-to-end encryption all messages received to and sent by the server will be encrypted via various algorithms and associated with private keys. This prevents any third parties from accessing user or other sensitive data. RBAC ensures that individuals associated with specific roles such as admin, user, player etc. can only access necessary data. Each individual will be assigned a role and the level of access within the system will be decided by this role. Finally, a honeypot acts as a virtual trap to attract potential attackers so we may identify and study them. This could prove to be quite beneficial in the long term as more vectors and actors of attack are identified.