

Draw It or Loose It

# **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 | 03/24/24 | Oleksandra Kondieieva | Initial prototype. Developed initial requirements, constraints, and recommendations for the Gaming Room web-based game. |

**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 wants to expand its player base by developing a web game based on its current mobile game, *Draw It or Lose It.* At present, this game is available only on the Android platform. *Draw It or Lose It* is loosely similar to the 1980s television game Win, Lose or Draw, where teams compete to guess what is being drawn. To facilitate the development of the web-based version of the gaming app that serves multiple platforms, The Gaming Room needs our help in streamlining the development.

As for proposed solutions, we can offer cross-platform software, a user-friendly interface, performance optimization, time-based rounds, a scoring system, and leaderboards. The cross-platform game will be developed and adapted to the specifications of each platform. With a user-friendly interface, we can make a game intuitive for first-time players and ensure the seamless integration of the current app users. Performance optimization will help to coordinate with an extensive library of stock drawings and guarantee stable image rendering speed. Time-based rounds lasting 1 minute will help keep track of a score and make the game more challenging. Scoring systems and leaderboards will help motivate other players and keep them invested.

## Requirements

Here are some of the most essential requirements for the future game:

1. The web game should serve multiple platforms.

2. The game should render images from a library of stock drawings as clues.

3. The game has to consist of four 1-minute rounds.

4. Drawings must be rendered at a steady rate and fully completed at the 30-second mark.

5. If the team does not guess the puzzle before time expires, the remaining teams have to have an opportunity to offer one guess each to solve the puzzle with a 15-second time limit.

6. The game must have the ability to have one or more teams.

7. Each team must have multiple players assigned to it.

8. Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.

9. Only one game instance can exist in memory at any given time. This can be accomplished by creating unique identifiers for each game, team, or player.

## [Design Constraints](#_2et92p0)

Some of the design constraints that might be present in this project are:

*1. Multiplatform compatibility****.***  
 The cross-platform requirement is considered a constraint on multiple levels. First, the technologies for developing a game must support cross-platform functionality. The user interface has to be adapted specifically to each platform. Performance should be optimized and equalized across all platforms, including specifications. Testing will also require different platforms to ensure smooth work and speed of rendering.

2. *Image library incorporation*

The game needs to use images from the library that are rendered at a steady rate. The development team needs to incorporate the library most efficiently since changes in speed rendering or any problems with the library will lead to a negative user experience.

3. *Multiple team and multiple players in a team options*

The game logic should accommodate multiplayer management. The game architecture and interface should support it. Unique names and ids should be created and tracked.

4. *Performance speed*

Performance optimization is essential because picture rendering is the base of this game. The developer team must ensure that rendering images from the library, player interactions, the number of players (the max has to be limited), and other factors don’t influence the overall performance. This also has to be checked for every platform this game is available for.

## [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 diagram of The Gaming room represents seven classes – ProgramDriver with the main() method, SingletonTester with the testSingleton() method, Entity, Game, Team, Player, and GameService. The ProgramDriver and Singleton tester have a directed association relationship. The control flows from ProgramDriver to SingletonTester. The Entity class is a base class that introduces common attributes and behaviors. The Game, Team, and Player are the child classes of the Entity; thus, they inherit all the attributes and methods. GameService and Game, Game and Team; Team and Player classes have association zero-to-many relationship. For example, the team can have zero or multiple players. Each child element of the Entity, in addition to inheriting common attributes and methods, also has its extended list of methods and instances.

This example shows the basic principles of object-oriented programming. These four concepts are *encapsulation, inheritance, portability, and polymorphism*. ***E***ncapsulation is presented by private instances of the Entity class, such as *id* and name. Encapsulation helps to hide the components from accessing by others directly. ***I***nheritance is presented by the Entity (parent) class and the Game, Team, and Player (child) classes, inherited from the Entity class. The inheritance helps establish the relationship between the classes and keeps the code clean without rewriting the same code block. ***P***ortability can be seen in the Entity class, for example. This code can be used in different programs, and in case there is any problem in the code, it can be fixed in the Entity class, which makes the troubleshooting effective and efficient. ***P***olymorphism is presented by using the overloaded methods in the Entity class. For example, there are two overloaded constructors with and without arguments. Polymorphism helps to use the inherited methods to perform different tasks.

"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** | pros:  -stability;  - security;  - reliability  -integration with Apple ecosystem;  cons:  -scalability concern (if the amount of users increases significantly);  -limited server tools  - dependency on the Apple ecosystem  - costly; the licensing price is integrated into the hardware price | pros:  - stability;  -security;  -flexibility;  - may run on various hardware architectures;  -customization;  - open-source  - free of cost licensing (some vendor distributions may require payment)  cons:  - possible hardware compatibility issues;  - fragmentation;  - lack of support from the developers, since it is open open-sourced and everybody can contribute | pros:  - ease to use;  integration with Microsoft system  - compatibility;  -support;  cons:  - vulnerable security system; - dependency on the Microsoft ecosystem; -require more resources that may influence performance and possible scalability  - costly; the price depends on a licensing model | pros:  -integration with other apps;  - portability;  - possible instant updates;  - integration with the phone’s features.  cons:  - limited resources;  - potentially vulnerable security system;  - compatibility issues for different mobile operating systems;  - dependence on mobile internet  --might be high in cost if develop an app for different devices, testing, and constant maintenance;  - not typically used for hosting bu can be used as clients |
| **Client Side** | In order to ensure that the application is compatible with all web browser platforms and mobile devices, it is required from the application development process to incorporate continuous and cross-browser testing, responsive design, performance optimization depending on the platform specifics, and adaptation of available features on every platform. | | | |
| - Developer-friendly environment; - average development time frame;  - costly; the licensing price is integrated into the hardware price - requires platform-specific knowledge | - Developer-friendly environment;  - average development time frame;  - it takes some time and expertise to learn how to develop using Linux without prior experience.  - free of cost licensing (some vendor distributions may require payment) | - Developer-friendly environment;  - average development time frame;  - requires platform-specific knowledge  - costly; the price depends on a licensing model | - Developer-friendly environment;  - takes more time than focusing on one platform; - requires specific knowledge, but the cross-platform framework may be used  --might be high in cost if develop an app for different devices, testing, and constant maintenance; |
| **Development Tools** | - MacBook with macOS;  - Swift;  - C++  - Xcode (IDE)  - dependency manager;  -Xcode is free for all Apple users; | - C++/Python/JavaScript;  - Visual Studio Code/ Eclipse;  - Git;  - IDE can be used for free but other type of the subscription will require a payment | - C++;  - Visual Studio (IDE);  - Windows SDK;  - .NET framework  - IDE can be used for free but other type of the subscription will require a payment | - Java/Swift;  - Android Studio/Xcode;  - apps for mobile testing;  -multiple development teams might be required for iOS and Android  - Android Studio is free and Xcode is free for Apple users |

## 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 that The Gaming Room utilize the Linux platform, which is stable, secure, and flexible. Cross-platform development frameworks, like Unity, will help create a game for multiple environments efficiently.
2. **Operating Systems Architectures**: Linux architecture consists of an application, shell, Linux kernel (provides necessary services to operate the hardware), and hardware. Hardware includes CPU (responsible for executing instructions), RAM (store data and programs used by CPU) and input-output devices (keyboard, monitor, etc). Linux kernel is a base for other software that works on top of it. The kernel is responsible for memory, device, and resource management. Shell is an interpreter that translates the command from a keyboard into a language that a kernel can understand. Applications include all the programs a user runs on top of an architecture.
3. **Storage Management**: I recommend using a serverless architecture to approach storage management. This model will ensure that the cloud provider manages the allocation and server’s provisioning. The proposed approach will reduce the cost of maintaining the game because the price depends on the memory required and is also execution-based. The serverless architecture is also beneficial in the seamless scaling in case the Gaming Room would like to expand the game’s library or accommodate more players.
4. **Memory Management**: With the help of virtual memory management, we will ensure that the app has sufficient addresses to run the game smoothly. Memory allocation will be helpful in dynamically loading picture libraries and storing data structures. Caching will help store some information that will reduce the load time. Memory paging will aid with allocating more memory for storing information. Memory compression (memory pages in RAM) will be helpful in mitigating the pressure on physical memory.
5. **Distributed Systems and Networks**: I recommend using the client-server architectural pattern. This architectural pattern consists of two components – clients and servers that take clients' requests and provide service based on these requests. This is particularly helpful for a web-based application that must be able to run on multiple platforms. Since API and standard protocols can be used for communication with a server, clients can be developed using different programming languages and adapted for a specific platform. Client-server architectural design patterns also help to optimize network usage by transferring only relevant information between servers and clients. This pattern also helps manage the load of users; if reaching a specific number of clients, the extra load can be transferred to a different server. The client-server pattern aids with separating the user interface from the game logic.
6. **Security**: I recommend using secure protocols to encrypt data transmitted. It will be helpful to implement a strong authentication and verification procedure, like two-step verification. Implementing input validation and rate limiting will help to prevent DDoS attacks. Timing out a user from entering credentials after a couple of unsuccessful attempts and immediately notifying them by email will aid in ensuring security. Linux offers a Secure Boot feature that prevents the execution of unauthorized code. Linux also implements role-based access control to make sure that each user has their own account and permissions. Regular security updates will be beneficial in developing and using the future game.

REFERENCES:

Bashir, F. (2019, July 13). *What is serverless architecture? what are its pros and cons?*. freeCodeCamp.org. <https://www.freecodecamp.org/news/what-is-serverless-> architecture-what-are-its-pros-and-cons/

JavaTPoint. (n.d.). *Linux memory management.* <https://www.javatpoint.com/linux-> [memory-management](https://www.javatpoint.com/linux-memory-management)

*Macbook vs. windows: Which operating system is better? comparison ...* Woodcessories. (2023, March 15). <https://en.woodcessories.com/blogs/news/macbook-vs-windows-> comparison

Oishi, A. Z. (2023, November 20). *Architecture of Linux Operating System*. LinuxSimply. <https://linuxsimply.com/linux-basics/introduction/architecture-of-linux-operating-> system/#:~:text=The%20architecture%20of%20Linux%20is,and%20creates%20the %20whole%20system.

Patnaik, M. (2023, Nobember 9). Advantages of Operating System: Unveiling the Core Pillars. *AlmaBetter*. <https://www.almabetter.com/bytes/articles/advantages-of-operating-> [system](https://www.almabetter.com/bytes/articles/advantages-of-operating-system)

Red Hat. (n.d.). Logical Volume Manager (LVM). *Red Hat Enterprise Linux 5 Deployment Guide.*  <https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/5/html/> deployment\_guide/ch-lvm

Zhang, W. (2021, February 4). *Linux Kernel vs. Memory Fragmentation (Part I)*. PingCap. <https://www.pingcap.com/blog/linux-kernel-vs-memory-fragmentation-1/>

*Working with UEFI secure boot.* Oracle Help Center. (2024, January 4). <https://docs.oracle.com/en/operating-systems/oracle-linux/secure-boot/sboot-> OverviewofSecureBoot.html