

Draw It or Lose It

# **CS 230 Project Software Design**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_heading=h.gjdgxs)

[**Table of Contents,**](#_heading=h.30j0zll) [**Document Revision History**](#_heading=h.3znysh7), [**Executive Summary**](#_heading=h.2et92p0),[**Requirements**](#_heading=h.tyjcwt),

[**Design Constraints**](#_heading=h.1t3h5sf)  [**2**](#_heading=h.30j0zll)

[**System Architecture View and**](#_heading=h.4d34og8) [**Domain Model**](#_heading=h.2s8eyo1) [**3**](#_heading=h.4d34og8)

**The Gaming Room UML Diagram** [**4**](#_heading=h.17dp8vu)

[**Evaluation (Chart)**](#_heading=h.17dp8vu) **5 - 6**

[**Recommendations**](#_heading=h.26in1rg) **7**

**References****8**

## [Document Revision History](#_heading=h.lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 03/17/24 | Mikayla-Joy Botha | Completed the Executive Summary, Requirements, Design Constraints, System Architecture View, Domain Model, Evaluation, Recommendations, References |

## [Executive Summary](#_heading=h.35nkun2)

Creative Technology Solutions (CTS) has appointed me as a new technology consultant on behalf of The Gaming Room, whose current Android app-only game "Draw It or Lose It" necessitates the development of a cross-platform web-based game. The game will require teams to compete by trying to guess the outcome of a drawing within a set of four rounds, each a minute long, and the images are generated from a stock drawing library.

## Requirements

* Multi-Team Support: The game should accommodate the simultaneous participation of multiple teams.
* Team Management: Multiple individuals may be assigned to a single team.
* Unique Names: To avoid confusion and help users check name availability easily, it's best for game and team names to be distinct.
* Singleton Instance: To ensure that resources are well allocated for optimization, only a single game instance should be found in memory at any time.

## [Design Constraints](#_heading=h.1ksv4uv)

Creating this game application in a web-based system involves certain limitations throughout the design process. Primarily, it is essential to guarantee immediate synchronization and communication between various clients accessing the game. These developments must include reliable network protocols to ensure a smooth gaming experience and efficiently manage potential incoming delays. Furthermore, it is imperative to establish and enforce all security protocols to protect user data from unwanted access or possible manipulation. Scalability is another critical factor to think about, as the "Draw It or Lose It" game must be able to handle an increasing number of people without sacrificing performance or speed.

## 

## 

## [System Architecture View](#_heading=h.44sinio)

For The Gaming Room's "Draw It or Lose It" game application, the system architecture consists of significant java classes such as Entity.java, Game.java, GameService.java, Player.java, ProgramDriver.java, SingletonTester.java, and Team.java. As a main entry point, the ProgramDriver class is the primary method used to execute the program. The ProgramDriver class communicates with the SingletonTester class to verify that the singleton pattern has been implemented. Next, the Entity class establishes a foundation of shared encapsulated characteristics like attributes (id, and name) and behaviors (Entity(), Entity(id: long, name: String):, getId():, getName():, and toString()) that are inherited by other classes. The GameService class is responsible for overall game functionality, including generating unique identifiers for game instances and creating, retrieving, and supervising all games, teams, and players' information. The Game class, connected to the GameService class, deals with individual games, manages teams affiliated with each game, defines methods for adding teams, and transforms any game information into string format. Thus, the Game class encapsulates particular attributes and behaviors for every game instance. Any game player's data, for example, their unique identifier and name, gets stored in the Player class. Lastly, any players will be listed in the Team class, which organizes which teams they belong to and offers the ability to add new players to any of the teams. Overall, this system coordinates the development and administration of games, teams, and players by utilizing multiple classes to promise a seamless experience.

## [Domain Model](#_heading=h.2jxsxqh)

In the UML class diagram provided below, a package named "com. gamingroom" is found in the "Draw It or Lose It" game domain model. This package contains seven classes: ProgramDriver, SingletonTester, Entity, GameService, Game, Team, and Player. As previously discussed, the ProgramDriver is the program entry point for project execution and is the primary function. The SingletonTester class confirms that the singleton pattern has been implemented. The Entity class is basically a metaphorical container with a couple of shared attributes and behaviors that extend to different classes. Then, the GameService class works like a subsystem in charge of games and connects to the Game class, which connects to the Team class and then to the Player class through associations. It's important to mention that because the GameService class incorporates the singleton design, it conveys that only a single instance will be found in the memory. The game class works with various teams, securing things for each specified team. The team class controls a list of players, and the player class deals with each player's personal and sensitive information. An associative relationship within the UML is between the GameService and Game classes, which is depicted by the solid line connecting the two classes. This means that the GameService class manages game objects for numerous Game instances. When it comes to inheritance, an example shown in the UML below is between the Entity class, the Game class, the Team class, and the Player class, which is depicted by the empty arrowhead pointing from the subclasses to the superclass. Below, you can see that the Entity class is located above the game, team, and player class, meaning that the Entity class is the superclass while those mentioned below are the subclasses. Because subclasses such as the Game, Team, and Player classes all inherit from the Entity superclass, any attributes and behaviors from the Entity class will be passed onto the subclasses, enabling code reusability. Overall, the design of this domain model demonstrates object-oriented programming concepts with the usage of encapsulation, inheritance, and association to satisfy the project requirements.

**"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](#_heading=h.z337ya)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac is known for its strong performance and dependability, making it an excellent choice for hosting web-based software applications. With its Unix-based architecture, this operating system promises reliability and security. Furthermore, it is compatible with popular development frameworks such as Node.js and Ruby on Rails, making software development easy. On the flip side, Mac servers are prone to cost more when compared to their Linux counterparts. | Linux is recognized for its reliability, impressive security measures, and adaptability, which makes it a popular option for hosting web-based software applications. Linux offers cheap solutions and a large support community thanks to its open-source nature. Development tools like Apache and Nginx can be downloaded quickly, but Linux operations can be challenging for novices. | With its user-friendly interface, Windows provides a comfortable platform for hosting web-based software applications. Its seamless integration with Microsoft Tech, including .NET and ASP.NET, is a boon for developers. However, Windows servers may have vulnerabilities, and the costs associated with licensing can be significant, especially for large-scale deployments. | Mobile devices supply the convenience of being both portable and easily accessible, allowing users to access web-based applications when they are on the go. Whenever creating apps for mobile devices, it is essential to focus on different operating systems like iOS and Android. Each platform has its development frameworks and tools that should be considered. Beyond that, mobile devices generally have less processing power and network connectivity when compared to desktop or server systems. |
| **Client Side** | There are a few points to consider when it comes to supporting different clients on Mac. Compatibility with different macOS versions and appropriate development tools like Xcode are essential. Experience in programming languages like Objective-C or Swift is also paramount. Apple software and hardware might be more expensive than other platforms, so budgeting is crucial. | Being involved with different clients on Linux demands considering factors like compatibility with unique versions of Linux, the accessibility of development tools like GCC and JetBrain, IDEs, and competency in programming languages such as C, C++, and Python. Linux presents reasonable solutions, yet the broad spectrum of variations may cause problems when it comes to software compatibility. | For optimal compatibility with all versions of the Windows operating system and support a range of clients, it's crucial to have access to development tools such as Visual Studios and practice programming languages like C# and . NET. Window tools can be expensive due to their development tools' wide range of capabilities. The complex structure of the Windows operating system could require extra testing to ensure smooth functioning. | Specific development frameworks such as Swift for iOS and Java or Kotlin for Android are required to support different mobile devices. This includes maintaining compatibility with different screen sizes and resolutions. The costs related to mobile app development can vary depending on the chosen platform. |
| **Development Tools** | Programming languages and tools commonly used for Mac software include Xcode, which supports Objective-C and Swift programming languages. Moreover, development frameworks such as Cocoa and Cocoa Touch are frequently used. | Programming languages and tools frequently utilized for using software on Linux include GCC for C and C++ development, Python for scripting, and IDEs like JetBrains or IntelliJ for Java development. | Programming languages and tools often used for deploying software on Windows include Visual Studio, which supports C# and . NET. Development frameworks like ASP.NET are also regularly used. | Programming languages and tools for deploying software on mobile devices include Xcode for iOS development along with Swift or Objective-C, Android Studios for Android development including Java, and other cross-platform development frameworks like React Native and Flutter. |

## Recommendations

* Operating Platform:
  + After carefully evaluating the project requirements and considering factors such as scalability and cost-efficiency, it seems that Linux would be the best choice for improving the "Draw It or Lose It" game as opposed to other operating platforms. A Linux operating platform will allow my client, The Gaming Room to expand their "Draw It or Lose It" game in a way that will meet a large majority of project requirements so that it will be adaptable with hosting web-based applications.
* Operating Systems Architectures:
  + A Linux operating system is built on a Unix-like model, which provides security, reliability, and versatility for the architecture. An example Linux operating system, such as Ubuntu, offers comprehensive support for server environments, making it ideal for hosting web-based applications.
* Storage Management:
  + An appropriate store management system for the Linux operating platform and servers like ext4 or XFS may promote efficient storage when hosting web-based applications. Plus, implementing things like RAID can limit data redundancy and fault tolerance.
* Memory Management:
  + Memory management techniques such as virtual memory, page caching, and swapping allow Linux operating platforms to maximize memory use when hosting web-based applications. These techniques prevent memory leakage and system failure to secure optimum memory allocation.
* Distributed Systems and Networks:
  + With the establishment of software architectures such as microservices or containerization, the client has the "Draw It or Lose It" game to communicate between various platforms. Examples of these platforms include Docker and Kubernetes, which can help facilitate control over the distributed software and the networks that connect the devices, maintaining scalability and steady connectivity.
* Security:
  + On Linux platforms, user information is protected by incorporating security features like encryption, user permissions, and firewalls on and between various platforms. Also, with the integration of HTTPS protocols and private communication channels, security features may be further strengthened between different platforms.

**References**

* Docker. (n.d.). What is Docker? Retrieved from https://www.docker.com/what-docker
* Kubernetes. (n.d.). What is Kubernetes? Retrieved from https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/
* Linux Foundation. (n.d.). What is Linux? Retrieved from https://www.linuxfoundation.org/what-is-linux/
* Soltesz, S., Pötzl, H., Fiuczynski, M. E., Bavier, A., Peterson, L., & Elf, M. (2007). Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. ACM SIGOPS Operating Systems Review, 41(3), 275-287
* Stallings, W. (2014). Operating systems: Internals and design principles. Pearson Education.
* Ubuntu. (n.d.). Ubuntu server. Retrieved from https://ubuntu.com/server
* Shaw, M. K., & Garlan, D. (1996). Software architecture: Perspectives on an emerging discipline. Prentice Hall.
* Saltzer, J. H., Reed, D. P., & Clark, D. D. (1984). End-to-end arguments in system design. ACM Transactions on Computer Systems (TOCS), 2(4), 277-288.
* Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating system concepts. Wiley.
* Anderson, T. E. (2008). The performance of spin lock alternatives for shared-memory multiprocessors. ACM Transactions on Computer Systems (TOCS), 15(4), 348-382.