

**Draw It or Lose 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 | 05/21/2024 | Suraj Hamal | Executive summary, Design constraints, domain model with its description and UML diagram for the web-based game application-draw it or lose it, for our client The Gaming Room. |

## [Executive Summary](#_sbfa50wo7nsh)

Our new client, The Gaming Room, wants us, Creative Technology Solutions (CTS), to develop a web-based game that serves multiple platforms which is based on the version of their Android Game, Draw It or Lose It. The game is similar to the 1980s television game Win, Lose or Draw, where teams compete to guess what is being drawn. In our application, CTS will render images from a large library of stock drawings as clues rather than a player drawing images on an easel for the team members to guess the puzzle. CTS will create a gaming web application where multiples teams and multiple players are allowed to participate with their unique name which will be checked upon creation. The web-application will use unique identifiers for each instance of a game, team, or player to ensure only one instance exists in the memory at any given time.

## [Design Constraints](#_2et92p0)

* The web-based game application should be compatible across multiple platforms as the Gaming Room aims to reach a wider audience from its current android app.
* The application should render images from a large library of stock drawings as clues at a steady rate and are fully complete at the 30-second mark. If the team does not guess the puzzle before time expires, the remaining teams have an opportunity to offer one guess each to solve the puzzle with a 15-second time limit.
* The game should support multiple team’s participation where multiple players can participate in each team competing to guess the puzzles.
* The application should check for the game and team name in the system before allowing users to create one ensuring the uniqueness.
* One instance of the game should always exist in the memory at any given time, which can be achieved by creating unique identifiers for each instance of a game, team, or player.
* The web-based application should be developed based on the client’s Android app for a similar and familiar user experience for their customer base.

## [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 below visually represents the design and implementation of the Draw it or lose it web-based game application. Each class and its characteristics are designed based on the object-oriented programing which are described below.

**Entity Class:** The Entity class is the parent class or also known as super class for its child classes or also known as subclasses Game, Team and Player which holds all the common attributes and methods. One of the principles of OOP known as Inheritance is used by Game, Team, and Player classes as they inherit the methods and attributes from the Entity class. By adopting the **inheritance**, we can reuse, extend, and override the methods and attributes of this Entity parent class according to our need.

**GameService Class:** The GameService class uses the singleton pattern so that there is only one instance of the class at any given time and has global access to it. The GameService class is the main engine of the game which has attributes and accessors methods needed for the functionality of the game engine. The GameService class also implements the **encapsulation**, one of the principles of OOP, to hide its data components and restricts its access to the external entities.

**Game Class:** The Game class inherits from the Entity class, has zero to many relationship with GameService class and Team class (meaning a GameService can have zero to multiple games), uses the iterator pattern in its method to check for the team’s name before adding a new team, and uses **polymorphism** to override the toString() method inherited from Entity class for its own use.

**Team Class:** The Team class inherits from the Entity class, has zero to many relationships with Game class and Player Class (meaning zero to many players and teams can be added), uses the iterator pattern in its method to check for the player’s name before adding a new player name, and uses **polymorphism** to override the toString() method inherited from Entity class for its own use.

**Player Class:** The Player class inherits from the Entity class, has zero to many relationships with Team class (meaning zero to many players can be added to the team), uses the iterator pattern in its method to check for the player’s name before adding a new player name, and uses **polymorphism** to override the toString() method inherited from Entity class for its own use.

**ProgramDriver Class:** The ProgramDriver class is the driver class of the application which has the public main method to start-up the application. It uses the SingletonTester class to check if there is only one instance at any given time.

**SingletonTester Class:** The SingletonTester class is simply used by the ProgramDriver class to test to make sure there is only one instance of the game exists in the memory at any given time.

**"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** | MacOS discontinued its Unix-like server a couple of years ago. Therefore, we cannot use its platform as our server for game application.  -------------------------  However, incompatible with other operating systems and file system with potential higher cost for our client in comparison with Linux, Windows, and mobile devices. | Build upon Linux kernel with software packages and utilities called Linux distributions, Free open-source operating system with extreme stability and security than other OS, compatible with most file formats.  -----------------------  However, it is not very user-friendly as it requires more technical expertise than Mac or Windows. But good choice for our client as it is free, secure and compatible with other platforms. | Widely used and popular OS with user-friendly GUI with many versions available, compatible to most of the hardware and software available in the market, even offers versions for running servers.  -------------------------  However, it has security vulnerabilities, higher costs for our client for the server operating system, higher costs for its products or monthly subscription to utilize them. | | Increased mobility and portability with lower cost and maintenance, optimized to work under minimal power needs, runs mostly in touch screens, very convenient for users because of its attractive and easy to use interface  -------------------------  However, mobile devices have limited capability as it has the smaller screen size, limited processing power, and short battery life compared to that of desktops and laptops. |
| **Client Side** | Developing for the Mac clients will be expensive as it specifically needs the Mac software and hardware as well as the development team must be familiar with a Mac specific environment which may take more time. | Developing for the Linux clients will be cost effective as it is an open source and free platform, but it may take more time as it needs people with more technical expertise with the Linux system which is not very friendly compared to Mac and Windows. | Developing for the windows clients will be costly like Mac but provides a wide range of development tools, resources, and frameworks. Most of the developer are familiar with the overall operating system and its software will boost the game development faster than in Mac and Linux. | | Developing for the mobile clients may be time consuming because of different sizes, and types of device testing in multiple platforms such as Android and iOS. It may be costly too because of the platform specific development tools and licenses needed for the game to work on different platforms. |
| **Development Tools** | Development tools needed for this type of software can be  - Visual studio code  - Eclipse  - PyCharm  - Xcode  - Homebrew  - iTerm2  - Atom  -------------------------  Programming languages supported by Mac are Swift, C++, Java, Python, Ruby, JavaScript, and Objective-C. | Development tools needed for this type of software can be - - Visual studio code  - Eclipse  - Git  - PyCharm  -----------------------  Programming languages supported by Linux are C++, Java, Python, swift, Ruby, JavaScript, Kotlin, and Perl. | Development tools needed for windows are  - Visual studio code  - Gvim  - Git  - Microsoft Azure  - Jira  - IntelliJ IDEA  - Atom  -------------------------  Programming languages supported by Windows are JavaScript, C#, C++ Python, Java, VB.NET, R, and TypeScript. | Development tools needed for mobile devices are  - Android Studio  - Xcode  - AppCode  - Swift  - React Native  --------------------------  Programming languages commonly used by Android app development include Java and Kotlin, and IOS uses Swift and Objective-C with Apple’s Xcode IDE. | |

## 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**: The operating platform I recommend for our game application is Linux because of its cost-effectiveness, security, flexibility, reliability, and compatibility with other computing environments. As being the platform of choice for businesses and organizations around the world, Linux is open-source nature with customization flexibility, has wide range of applications and tools, has robust community support and resources, and highly scalable to handle large amounts of data and traffic. Its continued growth and usage as the most used server around the world make Linux an ideal choice for our game application.
2. **Operating Systems Architectures**: The Linux is an open-source operating system designed based on the principle of UNIX which consists of Hardware, Kernel, Shell, and System utilities and libraries.

**Hardware:** The hardware layer comprises the physical components of the computer such as hard drive, RAM, motherboard, CPU, and network interfaces.

**Kernel:** The Linux kernel is the core part of the OS as it interacts directly with hardware performing the low-level services to upper-level components. Linux is a monolithic kernel where the kernel directly controls device management, memory management, file management, and process management making the OS stable and fast.

**Shell:** The shell is an interface to kernel taking commands from the user and executing kernel’s functions in the Linux operating system. The shell forwards the user’s request to kernel for processing to perform various tasks such as running programs, managing files, and configuring the system.

**System Utilities and Libraries:** The system utilities perform the tasks given by the users and manage the Linux operating system. The system libraries provide the functions to the Linux operating system which interacts with the kernel and handles all the functionalities without the kernel privileges.

1. **Storage Management**: The appropriate storage management system for our game application would be the Amazon Web Services (AWS) cloud storage services. AWS offers pay-as-you-go cloud computing model where customers only pay for the storage they need and use which will be cost effective, reliable, scalable, and secure for our game application. The AWS provides many use cases to manage our data according to our need such as migrating applications to AWS, building data lakes, backing up to AWS storage, archiving to the cloud, processing data at disconnected edge locations, and modernizing our application development accelerating innovation.
2. **Memory Management**: The operating platform Linux manages the memory for our game application through its Linux kernel which is its core component of the operating platform. The kernel virtualizes the computer’s common hardware resources to provide each process with its own virtual resources to prevent and mitigate conflicts between different processes which can be ideal for our game application. The Linux kernel has the memory management unit (MMU) in its subsystems which manages the memory by performing different functions such as address translation, memory protection, virtual memory management, and memory segmentation.
3. **Distributed Systems and Networks**: The game draw it or lose will be deployed in a cloud-based serverless environment which is a distributed computing system. In a distributed computing system, processes and data storage is distributed across multiple devices or systems, clients and servers can both request and provide services to devices or systems in a network, resources such as power, storage, and networking are delivered over the internet and accessed on demand, and more devices or systems can be added to the network to increase processing and storage capacity. Our game application will use REST API and HTTP protocol to facilitate communication and data exchange between different devices and platforms.
4. **Security**: Securing our game server in Linux is important to protect our data from illegal access and security threats. We can adapt one of the guiding principles of the operating system - the principle of least privilege, which allows users, programs, or even the systems enough privileges to perform their tasks. We can prevent unauthorized access by implementing various security measures such as user authentication – authenticates users before accessing the system, access control – uses access control lists to give permission to certain users to perform certain tasks, and encryption – to protect sensitive data and prevent unauthorized access. We can defend our operating platform from potential risks by following the countermeasures such as intrusion detection system – to detect attempted or successful intrusions into the operating system, firewalling – to monitor and control the incoming and outgoing traffic in the system, virus protection – to protect system from viruses, and malwares, system updates and patches – keeping the operating system up to date with latest security patches and updates to protect it from the security threats