

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 | 07/15/2023 | Melissa Cianfarano | Updated Cover Page, Document Revision History, Executive Summary, Design Constraints, and Domain Model. |
| 1.1 | 07/26/2023 | M. Cianfarano | Updates to Document Revision History and Evaluation. |
| 1.2 | 08/05/2023 | M. Cianfarano | Updated requirement section. |
| 1.3 | 08/08/2023 | M.Cianfarano | Finalized updates to Document Revision History and Evaluation. |

## [Executive Summary](#_sbfa50wo7nsh)

Our client, The Gaming Room is requesting a web-based game that serves multiple platforms based on their current Android app game known as Draw It or Lose It. The multiple platform web-based game must mirror their already established Android app game and maintain the same rules and play. The game consists of four one-minute rounds of one or more multi-player teams guessing what image is being drawn. The images drawn are rendered from a large library of stock drawings and will be fully complete at the 30-second mark. If a team does not guess the image in the time limit, the remaining teams will have a 15-second time limit to take one guess. Each game and team name must be unique to allow users to check if a name is in use when choosing a team name and there may only be one instance of each game at a given time.

## Requirements

* Draw it or Lose it must run on multiple platforms based off of the current android app game.
* Must mirror preexisting game model.
* Able to flawlessly run through multiple instances at a time with unique player information stored.
* Four one-minute rounds of one or more multi-player teams guessing images being drawn.
* Easy access to a preloaded database of stock images to be fully drawn in 30 seconds.

## [Design Constraints](#_2et92p0)

There are a few design constraints for developing this web-based game application for our client, The Gaming Room. A game needs to be built which runs on multiple operating platforms. Currently it only runs on an Android app. This will require compatible universal languages which will be accepted by multiple types of platforms. The game must have the ability to support more than one game and one team containing multiple players at a time. The program will need to be able to avoid overloading issues. Each game and team require a unique ID and should also check whether the name already exists. This will require a database that will save team names which can be iterated through to ensure that there are no duplications. There will also only be one instance of each game at a time.

## [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 Gaming Room UML diagram visually represents properties and relationships between seven classes: ProgramDriver, SingletonTester, Entity, Player, Team, Game, and GameService. Each class is represented in a rectangle. The top portion of the rectangle contains bold print which represents each class name, the middle contains attributes, and the bottom contains methods. **Portability** is represented by multiple classes which can easily be used in other programs. **Inheritance** is demonstrated between GameService, Game, Team, Player, and Entity classes. These four classes inherit from the parent class Entity, meaning they will inherit elements from Entity. This is demonstrated by the open arrow from the child classes leading up to the parent. **Encapsulation** is an OOP concept where there are private methods that exist but cannot be accessed, which is represented by a negative sign in front of an instance. **Abstraction** is demonstrated by the SingletonTester class which means it ensures the program is running correctly, and it hides object implementation details. Team, Player, and Game demonstrate **polymorphism** as they contain different instances that are stored in specific places and will be assigned.

**ProgramDriver** – ProgramDriver class contains the main method and is used by the SingletonTester class. This is demonstrated by a filled in arrow which represents direct association. Association is a relationship where both classes need to communicate. In this case the SingletonTester holds a reference for ProgramDriver to use. This is further supported by the custom <<uses>> message.

**SingletonTester** – The SingletonTester class runs a test to ensure that there is only one instance of each game running at a time.

**Entity** – The Entity class demonstrates an inheritance relationship involving GameService, Game, Team, and Player classes. Elements from Entity will be inherited by each child class. Each child class has a solid line connecting them to represent association. Each contains a zero to many instance cardinality represented as 0...\*.

**GameService** – The GameService class contains the gaming method that contains a structure for the Game, Team, and Player classes. This contains the getter methods for each of these classes that it is associated with.

**Player, Team, Game** – Each of these classes holds information. Player class holds information about a player. Team class holds information about the team. Game class holds information about the game.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | **Pros**  • Can be used as a simple server which has extensive documentation for MacOS software.  • More readily available software with diverse implementation of technology, with fewer constraints.  • There is excellent technical support with an easy-to-use graphical user interface.  • Have proven security with anti-malware programs protecting from adware, spyware, and worms.  • Mac is the dominant platform compared to other types.  **Cons**  •Requires a MacOs system.  • Maintaining licensing is expensive, requiring payments and updates are provided solely through Apple.  • Limited hardware upgrades.  • Not great when relying on third party programs or customizations. | **Pros**  • Simple server configuration and access with use of a command shell.  • Stability.  • Cost effective being free with lowest hardware requirements.  • Open source with numerous libraries and utilities pre-packaged, yielding lowest technical constraints.  • Customizable security.  **Cons**  • Limited hardware support.  • Less preferred for web hosting services.  • Difficult to navigate with a long learning curve for a user with no prior experience.  • Limited software available when compared to windows.  • Migration issues from Windows to Linux as not all applications are supported.  • Susceptible to malware, spyware, and worms. | **Pros**  • Simple server deployment with flexible commands during configuration and a large range of hardware options.  • Intuitive GUI interface making it user friendly.  • Accommodates a high volume of people.  • Supports a lot of applications and third-party software.  • Able to run on more software with faster loading times.  •Extensive library for software support.  • Quicker to get easy patch and hardware updates. Easily detects security threats.  **Cons**  • Closed Sourced with poor technical support.  • Updates solely provided through Microsoft and can be costly to acquire and maintain licenses.  •Less secure than Linux and Mac.  •Limited customization. | **Pros**  • Runs mobile servers with low cost.  • Develop apps for phone applications and web-based applications interchangeably.  • Manages calls.  • User friendly and optimizes queries in the database.  • Highly compatible.  **Cons**  • Server needs to utilize cloud services or a physical server.  • Limited hardware upgrades and inability to handle large amounts of traffic.  • Poor security management with a server that can be tracked.  • Code inaccessible to user. |
| **Client Side** | • Costs are comparable to a Windows OS and can vary depending on users’ needs as they are not open source.  • Prior knowledge of MacOS is preferable as there is a learning curve for usage.  • Wide range of web browser support.  • Multiple developer friendly tools.  • Requires Apple products. | • Low costs, being open source.  • Customizable distribution, with developer friendly tools.  • Web supported browsers.  • Less loading time.  • Not very user friendly, prior experience preferred due to extensive learning curve.  • May be difficult to set up with few applications available.  • Prone to malware, adware, and worms. | • Can be costly depending on users’ needs, as it is not open source.  • Easy server development and administration.  • Intuitive GUI so minimum experience needed to use.  • Common so most users already have experience.  • Easy to understand for cross platform usage.  • Wide range of web browsers supported and developer friendly tools.  • Many time-consuming updates needed. | • Can be costly due to everchanging software, but comparatively cost effective compared to other platforms.  • Easy to use, experience not required with intuitive GUI.  • High availability of technical support.  • Lower loading times.  • Better portability.  • More time consuming for developers due to multiple OS and device types.  • Difficulty testing browsers and other environments.  Ability to see updates. |
| **Development Tools** | • Developers can utilize Xcode, Homebrew, and GIT.  • Some programming languages for MacOs include Swift, C, Python, JavaScript.  • Some IDE’s used include Xcode, Visual Studio, JetBrains IntelliJ IDEA, and Atom. | • Developers can utilize GNU Compiler Collection, Clang, Make, and Git.  • Some programming languages for Linux include C, C++, Python, Go, and Rust.  • Some IDE’s used include Visual Studio, JetBrains IntelliJ IDEA, Eclipse, VIM, and Emacs. | • Developers can utilize Visual Studio, .Net Framework, PowerShell, and Git.  • Some programming languages for Windows include C, Visual Basic, C++, and Python.  • Some IDE’s used include Visual Studio, JetBrains IntelliJ IDEA, and Eclipse. | • Developers can utilize Android Studio, Xcode, Flutter, and React Native.  • Some programming languages for Windows include Java, Kotlin, Swift, C, and JavaScript.  • Some IDE’s used include Android Studio, Xcode, Visual Studio, and InelliJ IDEA. |

## Recommendations

1. **Operating Platform**: To meet our clients’ needs, I recommend the project be implemented on a Windows OS. Windows is user friendly with it’s intuitive GUI. It has the tools needed to expand The Gaming Room’s Draw It or Lose it game to a multitude of platforms. Multiple IDE’s can be utilized to cover all game functionalities. Windows is known to have a strong gaming platform and can handle large loads of data without sacrificing speed. It allows for developer solutions that can cross perform on other platforms such as Android or Linux. The costs of maintaining a Windows based server are reasonable as well.
2. **Operating Systems Architectures**: Windows OS platform architecture has a kernel model and a user model. The kernel processes aren’t utilized by the user and deal with behind-the-scenes functionalities such as input, output, memory, networking, and hardware. The user model features an easy-to-use GUI. Windows has flexibility and allows for customization on a developer’s end to set up an environment to meet the clients’ needs. For maintenance there is a large amount of accessible documentation and data for troubleshooting needs. The OS can be used across multiple operating systems to help the program run smoother and have more fail safes.
3. **Storage Management**: Windows has a feature called storage sense which allows you to manage files on the hard drive. Windows also utilizes a disk management system to help with storage tasks. With built-in storage, it is possible to run multiple platforms. Another option is Microsoft Azure which is a cloud-based storage environment. Considering Draw It or Lose It is storing stock images to be used in a game, cloud-based storage would be an ideal method to utilize for this scenario.
4. **Memory Management**: Windows comes with memory management built in using a memory compression technique. Draw it or lose it will require a database for image archiving. Memory allocation allows for easy storage and faster loading speeds. Extra RAM reserved on the hard disk for extra storage. This helps by separating processes into smaller pieces which are loaded when required. All of these are ideal features for The Gaming Room’s requests.
5. **Distributed Systems and Networks**: To communicate between various platforms, we can accomplish this with distributed software and a network of connected devices. As mentioned before, cloud-based technology such as Microsoft Azure will allow distribution amongst networks and systems. This allows multiple connections in one instance, without one connecting disturbing the others. This type of distribution is vital in preventing a crash or delay the game and proves useful in employing multiplayer games across platforms. If one hub experiences a connectivity issue or outage, the game will continue. Draw It or Lose it will require extensive servers to support large amounts of data, therefore cloud-based technology will ease the burden.
6. **Security**: Windows OS comes with security features automatically built in. It has Windows Defender which can help with virus and threat protection. Windows is constantly updating its security software, reducing the rates of getting spyware, malware, or worms. There are multiple types of software readily available to help reduce the risk of security threats that can be added into the OS. Other preventions would be employing encryption methods to help ensure data protection for the client.