

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

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

[**Table of Contents**](#_heading=h.30j0zll)2

[**Document Revision History**](#_heading=h.3znysh7)2

[**Executive Summary**](#_heading=h.2et92p0)3

[**Design Constraints**](#_heading=h.tyjcwt)3

[**System Architecture View**](#_heading=h.3dy6vkm)3

[**Domain Model**](#_heading=h.1t3h5sf)3

[**Evaluation**](#_heading=h.2s8eyo1)3

[**Recommendations**](#_heading=h.3rdcrjn)5

## [Document Revision History](#_heading=h.3znysh7)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/06/2022 | Dante Lee | Wrote executive summary, design constraints |

**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](#_heading=h.2et92p0)

The Gaming Room needs to develop a web-based version of their game, Draw It or Lose It. The game renders images from a library of stock drawings, which are used as clues for the answer. Each game is 4 rounds, 1 minute per round. Drawings will render at a constant rate for 30 seconds until they are fully rendered. If the team doesn’t guess what the drawing is within that time, the remaining teams each get to provide one guess within 15 seconds. The game is currently only available on Android, so they want a web-based version to be built that can run on any device.

## [Design Constraints](#_heading=h.tyjcwt)

In order to create the environment correctly, there are a few requirements that need to be met.

First, the game should be able to run with one or more teams involved. Each team will have multiple players assigned to it. Game and team names must be unique, so that users can check whether a name is in use when selecting it. Only one instance of the game can exist in memory at a time, which will require each instance of a game, team, or player to have a unique identifier assigned to it. The game should run on all devices, which can be accomplished by creating a web-based version that will run on most devices such as MacOS, Linux, Windows, and mobile devices running iOS, Android (already built), and others.

## [System Architecture View](#_heading=h.3dy6vkm)

<null> redacted for page space

## [Domain Model](#_heading=h.1t3h5sf)

The driver class of the Gaming Room program uses the SingletonTester class to test that Game, Team, and Player class each only have one unique instance by their unique identifier. There should only be one instance of the GameService class at a time, while multiple instances of Team and Players may exist, but must each only have one per unique identifier. Using the Singleton pattern with a SingletonTest class along with this structure class will ensure that the GameService class only has one instance in memory, which has its constructor method set to private, only instantiated with its addGame method. If another instance’s creation is attempted, it should fail while being called if one already exists. Each Game, Team, and Player class will inherit from the Entity class, which holds the unique ID of each instance, along with the name of the instance. This will also make it possible and mandated that names are unique among Team and Player instances.

**"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.2s8eyo1)

| **Develop**  **Req.** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | **Characteristics:**  Simple command line and file system  **Advantages:**  Popular  **Weaknesses:**  Can’t run all software that others can | **Characteristics:**  Simple command line and file system  **Advantages:**  Cost-effective, much cheaper  **Weaknesses:**  Less people know how to use | **Characteristics:**  Can run a lot of different software  **Advantages:**  Widely used  **Weaknesses:**  Susceptible to certain vulnerabilities | **Characteristics:**  Very widely used, almost everyone has a phone  **Advantages:**  More people can use / Portable code  **Weaknesses:**  Certain devices will not be compatible |
| **Client Side** | **Cost:**  Cost will be similar to windows, licenses work differently  **Time:**  Decent amount of time required, similar to linux but more GUI  **Expertise:**  Some expertise is necessary, but not as much as linux | **Cost:**  Cheapest option; linux is open-source and free  **Time:**  Most time required  **Expertise:**  Most expertise required, knowledge of command line is mandatory | **Cost:**  Licenses will cost a decent amount of money, similar to Mac  **Time:**  Similar to Mac/ less than linux but depends on skill set  **Expertise:**  Some expertise necessary, different skill set from Mac/Linux | **Cost:**  Mobile should cost relatively the same to others as it is a flexible platform  **Time:**  Would require similar time to others, but time will be spent on different things  **Expertise:**  A very specific expertise is required in mobile development |
| **Development Tools** | **Tools**:  Visual Studio Code, Eclipse  **Languages**:  Swift, Python, Java, HTML/CSS/JS  Similar file system to Linux  Easier than Linux and Windows | **Tools**:  VS Code, Eclipse, CLI programs  **Languages**:  Python, C++, Java, HTML/CSS/JS  Similar file system to Mac  Harder than Windows, simpler file system | **Tools**:  Visual Studio, Eclipse, VS Code  **Languages**:  C++, Java, Python, HTML/CSS/JS  Easier to use than Linux, different file system than Mac/Linux | **Tools**:  Android/Swift  **Languages**:  Python, HTML/CSS/JS, |

## 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’d recommend that this product be first implemented on Windows machines, as Windows is the most widely used computer operating system for personal computers and business computers. A majority of the general public will have immediate access to their game, and streamers will be able to play it to further promote it. Windows, by name, is very GUI focused, more people and developers will be able to access the program and work on it.

1. **Operating Systems Architectures**:

Draw It or Lose It contains a large image library which needs to be smoothly and quickly rendered at precise times for opponents to have a fair chance against each other. These pictures should be held on a cloud server, which can be scaled according to traffic and popularity of the game. The client’s app will store all of the logic to execute the game functionality, while the server will store images and relay them to the client before each round, to keep it fast and not load too much at once, just in case someone quits before the end of the round we don’t end up using a bunch of data sending them multiple pictures for nothing. Since they will have to be connected to the internet in order to play against each other regardless, this should not affect play much.

1. **Storage Management**:

Images will be stored on a cloud server since the library is large, and will only be sent to the client machine before the start of each round, and not each game. With cloud, the image library can be seamlessly updated and the client will always be compatible with getting a random picture from the library since they are disconnected until the player plays the game. By not storing everything on the client, devices will not require much storage and the game will be able to load faster.

1. **Memory Management**:

Once an image is needed for the round and is sent to the client, it is stored in memory and can then be smoothly rendered without any delay or lag. Images are sent directly into memory, and only exist until the end of the round before being flushed out to make room for the next image’s rendering. In memory it will be important to ensure that both images load precisely at the same time, so this needs to be as efficient as possible.

1. **Distributed Systems and Networks**:

Since the client will want their game to run across many platforms, it is important to have a network that can help each communicate efficiently with each other. There should be a system in place to convert data into universally understandable commands and information across devices, which should back up any data that should persist after the game instance is over. Servers should be scaled, and be ready to be scaled proportionally to any increase in popularity of the game in order to avoid falling apart in the case of a spike in popularity.

1. **Security**:

Windows has built-in virus protection and security, but the tools used to develop *Draw It or Lose It* should have security in mind if an attacker were to penetrate the OS defense. The game should check for an update every time one is available to run, and should also not run without an internet connection to avoid certain attack strategies. Since it will be an online game anyways, this will be built into the design already. If local play were to be added to the game, it could be missing certain features that would require the internet.