

# **Draw it or Lose it**

# **CS 230 Project Software Design Template**

Version 2.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_l6ti7uoag22u)1

[**Table of Contents**](#_30j0zll)2

[**Document Revision History**](#_grjogdjh5fi8)2

[**Executive Summary**](#_sbfa50wo7nsh)3

[**Design Constraints**](#_2et92p0)3

[**System Architecture View**](#_ilbxbyevv6b6)3

[**Domain Model**](#_8h2ehzxfam4o)3

[**Evaluation**](#_2o15spng8stw)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 2.0 | 10/24/21 | Darleen Langan | Completed recommendations section |

**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 intends to develop their existing game, Draw It or Lose It, which is available on Android for cross platform application. Draw It or Lose It is based on an 80s television show wherein players in teams compete to guess a drawing in a stated time period. The game, Draw It or Lose It draws from a pool of stock images, rather than having players draw. The players are divided into teams and the game consists of four rounds. Each round lasts one minute, with the image rendering complete at the 30 second mark. Teams may then have a 15 second window to issue final guesses if the puzzle has not been solved at that time. The web-based design should provide for one or more teams to play, multiple players to be assigned to each team, unique game and team names, and finally, only one instance of a game may exist at a time.

## [Design Constraints](#_2et92p0)

Developing the game in a web-based distributed environment has several implications for the game in addition to the requirements outlined by the client. In order to provide for these requirements, Draw It or Lose It will be coded in JavaScript since this language has a high level of portability. Furthermore, the game may be developed as either a web-based game or be developed with inherited languages so as to make it compatible with multiple operating systems. In this way, Draw It or Lose It can be played on both Apple and Android based systems.

## [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 following UML class diagram presents the current architecture of the game. In this diagram, Entity is a superclass. The classes below it, Game, Team, and Player each inherits the attributes of the Entity class. The classes, GameService, Game, Team, and Player are related through aggregation, meaning each class contains an instance of the class to its left. Each class, hence, references an instance of another class.

This form of OOP (Object Oriented Programming) provides for the uniqueness of games and teams by requiring reference to the other classes. When an instance of a team is created, this object does the work within that class. That team instance then becomes an object in the Game class, then Game Service.

**"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** | Streamlined user interface. Simple commands to configure and change server and access data. | Cost friendly. Simple commands to configure and change access and server data. | Large variety of options for software. Most flexible. | Characteristics may vary based on product specifications and OS on each device. |
| **Client Side** | Because of simplicity inherent in Mac OS, there is a medium level of expertise and time required for implementation on Mac OS. | Cost is low due to access of software, but expertise level may be higher. Time requirements may also be higher due to limited expertise. | Popularity and variety of options results in low level of expertise, time, and cost. | Lowers time constraint for clients as access to development is broad. Expertise and time requirements are medium to high due to the varied specifications of products and OS. |
| **Development Tools** | Simplified development tools such as Unity, as well as Godot 2.0, Gdevelop, JavaScript. | Native programming tools such as GNU. Also, Unity, Godot. HTML5 browser-based programming also supported. | Many options available. Including: Java, JavaScript, C++, HTML5, SQL, Unity. | Numerous options that vary in supportability based on device specifications. May include Unity, Godot, JavaScript. |

## 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 Gaming Room requires its application, Draw It or Lose It to be functional across multiple operating systems. Because of this it is suggested that the client utilize a web-based platform, such as a web browser. This will enable users on various devices and using various operating systems to be able to use the application. Applications developed using HTML5 for example can harness many of the stunning features of traditional development and are also cross compatible. HTML5 apps are compatible across a multitude of devices, from mobile phones to PCs and can be used seamlessly between them.

While HTML5 is generally delivered solely through the user’s browser, a hybrid model may be used for Draw It or Lose It. This model would help ease some of the potential concern over internet connectivity in a game that is dependent on consistent speed of revealing an image between users. The game would be developed in HTML5 code that could be launched in a container app that would be OS specific.

1. **Operating Systems Architectures**:

The architecture of this web-based application is rooted in two processes, the client-side code and the server-side code. The client-side code is based in the user’s browser and responds to the user’s input. The server-side code is located on the server and it responds to the HTTP requests sent by the browser. For this application, all data is stored on the server side and is accessed by the user through input into the client-side code. The server-side code then produces all images, etc. that the user will see and delivers them via the web browser.

1. **Storage Management**:

For this architecture and to meet the requirements and restraints of the client, cloud-based storage is the appropriate type of storage management. This storage will enable the client to pay only for what is necessary currently while providing the option of expanding storage space easily in the future. Further, it will allow the client to avoid the complications and costs associated with developing its own servers, including hardware and design considerations, as well as the cost of the expertise required for implementing the server design.

1. **Memory Management**:

Memory management for this application requires that images be loaded to users on potentially different platforms at a fixed, consistent, rapid rate. The steady rendering of the images is essential for the game to be experienced fairly by players who may be logged in on different systems. As such, images should be stored on the cloud-based server and revealed to the user through server-side code. The use of cookies, and other forms of cache memory may help to ensure that potential differences in internet and processing speed do not affect the game play.

1. **Distributed Systems and Networks**:

A distributed system may be implemented to produce the desired goal of the client to create a game that can communicate across platforms. Users playing on different platforms will need to be able to access the game and even to play in the same game instance simultaneously. A distributed system can enable this outcome.

In a distributed system, multiple components may be located on different machines and/or servers and operate independently to produce a single cohesive product for the end user. Each machine/server/node works towards producing part of the end-product that the user sees as a cohesive system.

By developing Draw It or Lose It as a web based game in HTML5, the system will be inherently distributed. Some of the components include the operating system of the user, the web application of the particular user, the server where the game code is located, and the storage location(s) for the images and user data. By developing the game in this way, users with varying systems will be able to view the same end-product.

There are considerations, however, with respect to the effect of this development on game play. Slower internet connections or processing speeds have the potential to affect the ability to access the game, and/or for the game play to appear identical to different users (specifically with respect to the consistency and rate of the image rendering). However, utilizing short term memory storage features on the user side may help to alleviate these concerns.

1. **Security**:

Because the recommendation is to use cloud-based servers for this application, the heavy weight of managing security for the user data lies with the cloud storage company that is selected. It is highly recommended that The Gamin Room chose a cloud storage operation with a good track record for security. Servers are vulnerable to attack and the potential for user data to be stolen may be high if a less secure storage operation is selected. Servers should include firewalls, and intrusion detection systems, as well as being subjected to regular auditing.

The code for the game itself should also include security measures that are automated. Measures that can be employed to help stem cyber attacks include, Security Information and Events Management (SIEM) to address concerns that arise, utilizing a tool such as a CAPTCHA to prevent robotic phishing attempts, etc.