

# **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 | 07/18/2025 | Avrye Tompkins | Updating executive summary, requirements, design constraints, system architecture view, domain model, evaluation, and recommendations. |

## [Executive Summary](#_sbfa50wo7nsh)

Draw It or Lose it is based off a television game show Win, Lose or Draw circa 1980s. The idea of the game is for teams to create a drawing, then have to guess the drawing. The application moves that idea into a game application. This application will render images for a large library of stock drawings and clues. There will be four one-minute rounds, with drawings slowly being rendered in up until the thirty second mark, in which they will finish being rendered. If the original team does not correctly guess the prompt, the other team will have a chance to guess the prompt within 15 seconds. The Gaming Room does not know how to set up the environment. They also plan to create a web-based version of the app.

## Requirements

* The game will have at least one team with the ability to have more.
* Each team will have multiple players assigned
* Game and team names must be unique meaning if a name is taken, then another user can not select it. The system will need to check each name.
* There can only be one instance of the game running at one time. This can be completed by creating unique identifiers for the game, team, or player.

## [Design Constraints](#_2et92p0)

Some constraints could be having either too few or too many players. Not having enough images so the game does not feel boring. Some technical restraints could be not having enough memory or space for the number of people and photos. Another is properly loading in different photos each time, without repeats in the same rounds. Unsure of environment. Moving application to web based version. Checking to make sure that a unique identifier cannot be repeated.

## [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 first is that the whole “The Gaming Room” is in the folder of com.gamingroom, so all classes are within this folder. The first thing to note is that the SingletonTester has a usage relationship with the ProgramDriver. This is noted by a black arrow with the <<uses>> in the middle of the arrow. We also can note that the ProgramDriver, which runs the whole program also publicly contains the “main.” The SingletonTester class contains the public testSeinglton. These two have a disconnect from the rest of the classes.

The next note is that there are four classes which have a generalization arrow. The first class to look at is the entity class, which has the private id and name. It also contains a private entity class, with there also being public entity ids and name’s, a setter and getter for both names and Ids, and a toString. This allows for there to be only one instance of a name and Id to keep all information separate with no overlap. The Entity class inherits the Game class, Team Class, and Player class. The player class has public id and name, public toString. Players have a zero to infinite association with the Team class. This means that the class has infinite information for the players zero information. The Team class has a list of private players. Next the team class has a public Team id and name, adders for the Player class and a toString. The Team has an association to the Game class, with a zero to infinite association. The game class has a private team’s class list. It also has a public Game id and name, adder for team names, and a public to string. Finaly the game class is only connected to the GameService class. The game service class contains most of the information including name, id, getters, and setters. For the attributes, it contains all privately, games, nextGameId, nextPlayerID, nextTeamId, and service. These attributes contribute to keeping only one specific player id per team, one name per person, one id per game so there is no overlap. Next in the operations, we can see a private GameService class. Next are all public operations. This is Getinstance, addGame, getGame, Getgame, GetgameCount, getNextPlayer, GetNextTeamId. This sets a new player Id and gets the next round of IDs for multiple games. All this information once again is encapsulated in the com.gamingroom folder.

**"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** | <Evaluate Mac for its characteristics, advantages, and weaknesses for hosting a web-based software application.> | <Evaluate Linux for its characteristics, advantages, and weaknesses for hosting a web-based software application.> | <Evaluate Windows for its characteristics, advantages, and weaknesses for hosting a web-based software application.> | <Evaluate Mobile Devices for their characteristics, advantages, and weaknesses for hosting a web-based software application.> |
| **Client Side** | <Determine the software development considerations (cost, time, expertise) that are necessary for supporting multiple types of clients as they pertain to Mac.> | <Determine the software development considerations (cost, time, expertise) that are necessary for supporting multiple types of clients as they pertain to Linux.> | <Determine the software development considerations (cost, time, expertise) that are necessary for supporting multiple types of clients as they pertain to Windows.> | <Determine the software development considerations (cost, time, expertise) that are necessary for supporting multiple types of clients as they pertain to Mobile Devices.> |
| **Development Tools** | <Identify the relevant programming languages and tools (IDEs and other tools) that are used to build this type of software for deploying on Mac.> | <Identify the relevant programming languages and tools (IDEs and other tools) that are used to build this type of software for deploying on Linux.> | <Identify the relevant programming languages and tools (IDEs and other tools) that are used to build this type of software for deploying on Windows.> | <Identify the relevant programming languages and tools (IDEs and other tools) that are used to build this type of software for deploying on Mobile Devices.> |

## 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**: <Recommend an appropriate operating platform that will allow The Gaming Room to expand Draw It or Lose It to other computing environments.>
2. **Operating Systems Architectures**: <Describe the details of the chosen operating platform architectures.>
3. **Storage Management**: <Identify an appropriate storage management system to be used with the recommended operating platform.>
4. **Memory Management**: <Explain how the recommended operating platform uses memory management techniques for the Draw It or Lose It software.>
5. **Distributed Systems and Networks**: <Knowing that the client would like Draw It or Lose It to communicate between various platforms, explain how this may be accomplished with distributed software and the network that connects the devices. Consider the dependencies between the components within the distributed systems and networks (connectivity, outages, and so on).>
6. **Security**: <Security is a must-have for the client. Explain how to protect user information on and between various platforms. Consider the user protection and security capabilities of the recommended operating platform.>