

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

Matthew Lee Lewis

Matthew.lewis7@snhu.edu

## 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 |
| --- | --- | --- | --- |
| 1.0 | 07/17/22 | Matthew Lewis | This document provides a summary of the Gaming Room’s requirements, design constraints, and domain model for the “Draw It or Lose It” game app. |

## [Executive Summary](#_sbfa50wo7nsh)

The goal is to develop a web-based game application that runs on multiple platforms, titled “Draw It or Lose It,” which is currently available on Android only. There are to be multiple teams of several people each in the game playing four rounds for four minutes total (1 minute each.) Each round, a picture is pulled from a library, and one team guesses until the timer runs out. If they can’t get it, each opposing team is able to answer for 15 seconds.

## [Design Constraints](#_2et92p0)

* Only one instance of the game may run at once
* Needs one or more teams with multiple people on each
* Game and team names must be unique
* Needs to run on multiple platforms

## [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)

“Game”, “Team”, and “Player” are all children of “Entity” meaning they have an “is a” relationship with “Entity.” Basically, they are all types of entities, and all have “Entity’s” attributes as well as unique ones of their own. The relationship between “Team” and “Player” is “has a” because teams have players. One “GameService” can have multiple “Games” and each “Game” can have multiple “Teams” that are comprised of multiple “Players” each.

**"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** | The Mac OS X server provides a more affordable option but is also less popular than its alternatives. | The Linux OS has multiple distributions, each with its own server capabilities. It is very cost efficient, due to being open source but is also the least user-friendly option and you’d need skill with Linux. | The Windows OS’ functionality is often considered easier than the others. However, Windows is costly. | Mobile devices’ servers have pretty much zero cost but unfortunately do not provide nearly as much power as computers. |
| **Client Side** | Similar cost to windows and is not open source, meaning expertise with the device would be necessary. | Since it is not commonly used, a lot of practice, time, and experience will be necessary due to the open-source nature of Linux. | The cost would be in-line with the Mac’s cost on the client side. Similarly, it isn’t open source. | Extremely user-friendly so no experience or cost would be necessary. However, the different frameworks of various mobile devices could cause a timely issue. |
| **Development Tools** | Swift is the most common choice for Mac applications and numerous tools could be used including Atom. | Atom and Eclipse are commonly used for developing with Linux | Eclipse and Visual studio are exceptionally popular when developing with Windows. Visual Studio itself has a high compatibility with many programming languages. | There are a few options of what can be used, but most are written using Swift. |

## 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**: Windows is the top recommendation here due to its user-friendly UI and familiarity. Collaboration and compatibility with other organizations is made easier with Windows as well.
2. **Operating Systems Architectures**: The Windows GUI is very easy to use and the many capabilities it has as opposed to other operating systems will be very useful. The command prompt will likely be an invaluable asset and the high compatibility rate leading to more development tools makes Windows a must.
3. **Storage Management**: Cloud storage is a strong option with Windows’ built-in OneDrive as well as easily managed physical storage options as well. Windows also comes with a feature called Storage Sense, designed to help manage files stored on the physical drive.
4. **Memory Management**: Windows’ memory management has convenient upgrades and a minimum of 4GB for seamless usage. The Gaming Room will need a good database/library with many images. Memory allocation with windows allows for easy storage.
5. **Distributed Systems and Networks**: The Gaming Room will need strong servers able to support large volumes of players and backup power in case of outages. A game engine called unity could provide a good framework due to its high cross-platform compatibility and cost effectiveness.
6. **Security**: Windows does come with built-in protection via Windows Defender. It also provides clients with VPN security, Direct Access directory, etc. It can monitor threats in real-time, runs regular scans, and updates in real-time as well. Windows also has high support for all of the top tier security software if additional security is needed.