

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 | <01/23/24> | Dante Nardulli | Adding 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](#_sbfa50wo7nsh)

Creative Technology Solutions is creating a web-based version of The Gaming Room’s current Draw it or Lose it game. The game is inspired by the 1980’s TV show Win, Lose, or Draw. The web-based version will render images from a stock library as teams will compete against each other to guess the puzzle.

## Requirements

The software design addresses key requirements such as team and player management. It will also include unique names, and being restricted to only one instance of the game in memory.

## [Design Constraints](#_2et92p0)

**One or more teams:** The game needs to support one or more teams.

**Multiple Players per Team:** Each team needs to be capable of having multiple players assigned to it.

**Unique game and team names:** Game and team names need to be able to be unique and to allow users to check the availability when choosing a team.

**Single instance in Memory:** Only one instance of the game can exist in memory at any given point in time. To do this we will need to create unique identifiers for each instance of the game, team, or player.

Implications of these constraints include efficient team and player management, name uniqueness validation, and the implementation of a singleton pattern for the GameServices class.

## [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 UML class diagram consists of four classes: ProgramDriver, Entity, GameService, Game, Team, and Player The classes are connected as follows:

ProgramDriver<<uses>>GameService

GameService---0…\*-🡪Game

Game---0…\*🡪Team

Team---0…\*🡪Player

GameService---0…\*🡪Entity

Game---0…\*🡪Entity

Team---0...\*🡪Entity

Player---0…\*🡪Entity

**OOP:**

**Inheritance:** The Game, Team, and Player classes inherit from Entity class.

**Encapsulation:** The classes encapsulate their data and behavior, providing a clear interface for interaction.

**Singleton Pattern:** The GameService class follows the singleton pattern, ensuring only one instance in memory.

**"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** | MacOS is known for its Unix-based architecture, providing a stable and secure environment.  It has seamless integration with Apples ecosystem.  The hardware costs are one of its biggest weaknesses. | Supports a vast array of server-side technologies.  It has excellent stability and performance.  A weakness is its limited desktop adoption for end-users compared to Windows or macOS. | Commonly used in enterprise environments.  Windows has strong support for enterprise level applications.  A weakness is Windows may have a steep learning curve for those unfamiliar with the technology. | Good integration with back end with parse or firebase.  Direct access to device specific features.  Mobile devices have strict guidelines and approval processes for app distribution on the app store. |
| **Client Side** | With mac the development tools and licensing are quite expensive.  Development time may take longer due to device specific nuances.  Familiarity with Xcode is essential for efficient development. | Linux is open source which reduces licensing costs.  Compatibility testing may be required across different Linux distributions.  Knowledge of Linux server administration is very beneficial. | Windows server licensing may add to infrastructure cost.  Testing on different versions of windows may be necessary increasing time.  Develeopers should have expertise in .NET languages C# for development. | Development tools are generally free.  Although you need extensive testing on various devices and OS versions is necessary.  Developers need expertise in swift/objective-C for iOS and Kotlin/Java for Android. |
| **Development Tools** | Xcode for MacOS and iOS development.  Terminal for command line development.  Homebrew for package management. | Package managers like APT, YUM, or Zypper.  Code editors: Visual Studio, Vim, Sublime Text. | Visual Studio for .NET development.  Visual Studio Code for Cross-platform development.  Powershell for scripting. | Emulators for testing (Ios Simulator, Android Emulator).  TestFlight for Ios beta testing, Google Play Console for Android. |

## 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**: My recommendation is to go with Windows even though it is more expensive for licensing and software. You will save a lot of time during development. Using Windows comes with the use of Visual Studio and the many developer tools that it brings.
2. **Operating Systems Architectures**: Windows architecture is now Universal Windows Platform. This allows Windows 10 to run on multiple devices. It is designed to give a consistent user experience and apps can be distributed through the Microsoft store. Windows is a very flexible operating system as it supports many different architectures, i.e. 32-bit, 64-bit, as well as ARM architecture.
3. **Storage Management**: Windows is very effective because you can optimize disk space. This is done through the use of logical partitioning dividing the physical disk into partitions to separate the operating system, users, and system files. Windows also keeps the operating system and system files on separate partitions to make it easier to backup. Also, disc cleanup is very easy with Windows as it has a built-in function to do so. When it comes to storage space, it is actually managed pooling the physical drives together to be divided into virtual drives. This helps remove redundant storage leaving space for other files!
4. **Memory Management**: Windows utilizes Cloud storage which will be very beneficial to the end user by not having to store their app in a local file. This saves space and could even convince some people to download the app as it will not use their local storage.
5. **Distributed Systems and Networks**: This can be accomplished through client applications and developing platform specific applications, utilizing a central server component that manages game sessions. Using standard communication protocols like WebSocket for real-time bidirectional communication. I would also recommend deploying the server component on multiple servers with load balancing to distribute incoming requests.
6. **Security**: When it comes to security Windows uses encryption to secure data transmitted between clients and the server. You can implement secure user authentication mechanisms to protect user accounts and game sessions.