

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 3.0

## 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 | 7/14/2023 | Jonathan Thomas | All bracketed text replaced with appropriate responses. |
| 2.0 | 7/25/2023 | Jonathan Thomas | As it turns out, I need to do better at fully reading project rubrics and have accidentally already completed the sections that were not assigned until Project Two.  Added a sentence to the Mac section under Development Tools. |
| 3.0 | 8/10/2023 | Jonathan Thomas | Added recommendations for Operating Platform to be used by the Draw It or Lose It team and expounded on various features of the recommended OS. |

**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)

Because we are to develop a web-based game, we must decide on a programming language in which to build a game engine which will enable the game to be hosted on its own website or a website of our choosing. Preliminary research suggests that HTML5 is a strong contender as it is cross-platform and cross-browser capable. Of course, we need not re-invent the wheel – there are a plethora of HTML5 frameworks at our disposal in which a game engine can be easier created.

Next, because the game is an online-multiplayer game, we need to implement reliable, low-latency connections between all players and the server on which the game will be hosted. This will ensure that all players on each team can interact with the game simultaneously and in near real time.

## Requirements

The game must be hosted online via a web-based framework.

The game must have access to the image library from which the stock drawings are obtained.

The game will be team-based with each team consisting of multiple players.

To maintain fairness and to maximize the user experience, all players should connect to one server in order to sync their actions in a way that enables all players to interact with the game in near real time.

In a single game lobby, all teams must have unique names so that users may check whether a team name is currently being used.

Only one instance of the game may exist in memory at any given time and the game class should therefore be implemented with the singleton pattern.

## [Design Constraints](#_2et92p0)

The primary design constraint in my opinion is the ability to implement a low-latency line of communication from user -> network -> other users.

There are other design constraints regarding the implementation of player timers and timers which control the pace of image rendering during gameplay but these seem like simpler features to implement.

## [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 Game, Team, and Player classes all inherit from the Entity class which holds basic data such as id and name along with methods to access this data. In other words, the Game, Team and Player classes have a “is a” relationship with the Entity class.

From left to right, we see a series of “has a” relationships starting with the GameService class. So, GameService has Game(s) which has Team(s) which has Player(s).

**"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** | Mac does not prioritize gaming performance in any version of its architecture.  Mac UIs are well-known for being very user-friendly which may accelerate startup times for devs unfamiliar with the platform. | For the price of server hardware and an experienced dev, you too can have a reliable, responsive, and resilient game server.  Notoriously difficult to navigate for the inexperienced dev/user. | Not as expensive as Apple products but still expensive.  Generally user-friendly UIs to ease the learning curve.  For gaming, largely considered the “default”. | Some web-based designs can be hard to neatly translate to smaller screens or screens that may greatly vary in size.  Any web-based game that is playable on mobile devices would likely see better performance if shipped as a stand-alone mobile app. |
| **Client Side** | Mac products tend to be more expensive than most, if not all, other comparable hardware.  OS can be difficult to navigate if migrating from another OS.  Modifying your own hardware will often result in void warranties.  OS not optimized for gaming. | The primary tradeoff for the customizability of Linux systems is that the user must do nearly everything for themselves.  This includes ensuring that the OS build being used can provide a palatable gaming experience.  Free. | The “not as expensive as Mac” platform.  While the OS is not specifically designed around gaming performance, Windows is often the best bet for gaming performance.  OS is largely considered the default for anything gaming. | While not optimal for gaming performance, mobile devices have one important factor that is not present on any other platform: pure portability. |
| **Development Tools** | Although this was not always the case, development tools remain mostly consistent across all platforms.  This includes multiple IDEs such as those in the JetBrains family of IDEs and Visual Studio.  However, Mac is a user-favorite for the “creative” developer with their extensive suite of high-end editing and art creation software. | Other development tools that are generally cross-platform friendly include repository managers like git and GitHub. | While, historically, certain platforms have “favorite” languages in which to build applications (e.g., Swift for Mac or C# for Windows), by and large, desktop and mobile applications can be built in nearly any language. | The primary factor when choosing a platform for development purposes should be which platform the product will be deployed on. If the product is intended for multiple platforms as it is here, it may be wise to implement a multi-platform testing structure before deployment. |

## 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**: As mentioned above, Windows is largely considered the “default” for anything and everything gaming related – and for good reason. Windows platforms are generally user friendly and have a near endless supply of tools that can be used to ease development. Additionally, Windows is one of the, if not the most popular operating system in the world - meaning that developers working on the project will likely be familiar with their development environment and will not spend precious time learning the technical aspects of another platform.
2. **Operating Systems Architectures**: Windows provides some of the customizability of Linux while maintaining similar user safety features of Mac to prevent the average end-user from accidentally destroying their computer. In general, Windows provides the user with a graphical interface to store and interact with files and run software.
3. **Storage Management**: Windows provides an easy way for users to manage storage on their devices via File Explorer. The base page of the File Explorer window provides a visual representation of the current storage space being used vs free space. Alongside the visual representation are numerical values to precisely measure the amount of storage being used by files on the system.
4. **Memory Management**: Like many features, Windows provides developers the “rails” and mostly leaves the direction of development up to the user. Memory management is no different as it is built into Windows as a system utility. This differs from other platforms such as Linux which, while more customizable and therefore more powerful, relies on developers to build their own “rails” which ultimately leaves room for more vulnerabilities in the system.
5. **Distributed Systems and Networks**: As discussed previously, because the game requires the syncing of actions which originate from multiple users, all of whom may be on different platforms, the client-server communication network must be cross-platform capable and because all games are different, the server which syncs user actions, and the communication routes must be designed and implemented from scratch.
6. **Security**: While Windows comes with its own built-in security measures, because of the nature of cybersecurity, this may not be enough to fully protect our users. Additional security measures, including end-to-end encryption of all user data should be included to ensure total protection.