

The Gaming Room

# **CS 230 Project Software Design Template**

Version 1.0

## 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 |
| --- | --- | --- | --- |
| 1.0 | 04/16/2022 | Kentrell Edwards | Added information related to the software design |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room is working on a web-based game that will operate on various platforms. "Draw It or Lose It" is the game's title, which is presented exclusively accessible on Android. The goal of this game is for various teams of many persons to compete in four rounds of one minute each. One team guesses till time runs out when a photo is picked from a library of pictures. If the question is not answered, each opposing side member has 15 seconds to respond.

## [Design Constraints](#_2et92p0)

* One or more teams are required to participate
* Each team contains many persons
* Only one instance of the game may exist at any given moment, hence
* Game and team names must be unique to allow users to verify whether a name is in use or available.
* It must be able to run on many platforms.

These are the standards that must be observed while creating software and code. Even if this is only a game, application development must be considered. The Gaming Room would like to use this feature on any device. This means we already have it on Android and need to get it working on another device. Along with PCs running Windows, Linux, and Apple. We'll either have to rebuild the code in Swift for (Apple devices) or develop a way to inherit existing code from other languages and run it on different devices. We integrate multiple computer languages, for example, to produce more robust code.

## [System Architecture View](#_ilbxbyevv6b6)

Although this section is not required for these projects, it reminds us that other projects may require a detailed description of the system and subsystem architecture, including physical components and tiers. A logical topology of the communication and storage components is also necessary and should be described to appreciate the entire design.

## [Domain Model](#_8h2ehzxfam4o)

Entity is the glue that holds the Game, Team, and Player classes together. This means that Entity provides or inherits information from them all. In UML, we may show this using inheritance. As a result, all classes will share standard references like "name" and "id." The creature will be promoted to the superclass level. When we look at their relationship, we can see that Team and Player have a "has a" type of relationship. The game differs from GameService in that Game has a Team, whereas GameService has Games. In UML, we call this aggregation (HAS-A). When I say "has a," I mean that it's an instance of one class linked to an example of another class. This diagram shows how GameService refers to Games, Games to Tea, and Team to Player.

**"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)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac OS X server is a program that may be used on a Mac. According to Apple's website, Mac OS X Server is a mere $20, making it a low-cost solution. However, for these duties, Mac is not as popular as Linux or Windows. | Linux is notable for having a wide range of server-capable distributions. Linux Server would be a low-cost, open-source solution (with a wealth of information). Because few people are familiar with Linux, the server will need to be operated by someone who is. | Microsoft offers Windows Server as a service. According to Microsoft's website, an implementation may be costly, but it is fully functioning. Customers would be considerably more likely to install Windows servers since Windows is the most extensively used operating system. | Considering mobile devices don't always have the computing power of PCs, operating a fully working server on one isn't always the best option. Running servers on mobile devices is the most cost-effective solution in terms of cost, as it involves little to no setup. |
| **Client Side** | The cost would be comparable to a Windows installation because these operating systems are not open source. Time would be determined by competency, with someone with more Mac expertise having less time and less Mac experience requiring more time. | Considering Linux is a free operating system, the expense (if any) would be negligible. Because Linux isn't frequently used, you'll need a lot of time and experience and someone who understands their way around it. You'll also need to allow them plenty of time to work because Linux may be difficult even for seasoned users. | The cost would be comparable to a Windows installation because these operating systems are not open source. Time would be determined by competency, with someone with more Windows expertise requiring less time and someone with less Windows understanding requiring more. | The cost of mobile devices would not be a significant consideration. Because mobile devices are so simple to use, prior knowledge isn't often required. Since there are so many operating systems and mobile devices to work on, more time is necessary. |
| **Development Tools** | The most popular programming language for Mac applications is Swift. Swift is compatible with a wide range of IDEs, including Atom. | Eclipse and Atom, two popular IDEs for Linux, are extensively used. Eclipse is most commonly associated with Java, although it may also be used with other languages like C+. | Eclipse and Visual Studio are popular IDEs for Windows. HTML, C#, and JavaScript code can all be written using Visual Studio. | Although the development tools for iPhones and Macs are comparable, and iOS apps are frequently written in Swift, the interfaces for iOS and macOS are distinct. |

## 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**: After considering the level of expertise required and the relatively low cost, I recommend that The Gaming Room start the project using the Windows Operating System. The Windows operating system also comes with several functional integrated development environments (IDEs) that speed up and simplify the process.
2. **Operating Systems Architectures**: All Windows-based programs can use Windows services to display a Graphical User Interface (GUI), access system resources, and more. These applications include graphics and multimedia, communications, and internet services. These services can be accessed via a user account or a server.
3. **Storage Management**: Storage Spaces in Windows 10 allow us to add extra storage to the system, especially if there is the capacity for additional drives to be combined into a storage pool, which will improve speed and data security. It might also help us make the most of the storage we already have by reducing junk and using the cloud.
4. **Memory Management**: When we begin the project, we will produce and demand many files. We could construct those files from scratch, but there's a good chance that utilizing one of these game engines, which is a type of software development environment for game developers, will help things go much more quickly. Rather than hard-coding everything, the game engine may provide us with pre-built libraries and effects, saving us a lot of time. If we have our library database on the computer we use, it will be easy to organize these files.
5. **Distributed Systems and Networks**: Although I recommend starting with window-based programming, I know that we will eventually need to support all platforms, regardless of operating system or device. Using a cross-platform game design tool might be quite beneficial to this project. One of the most popular and cost-effective cross-platform game engines, Unity can run on Windows, Linux, Android, iOS, and other platforms. It's also critical to choose the correct server. Because many players may be playing the game simultaneously, it's vital to pick a network and server that can handle the high demand at all times.
6. **Security**: Security software is included with Windows, as it is with all other operating systems. Although this built-in application may perform well, if you are willing to pay for it, there is other software available to increase the system's overall security, as we will be dealing with sensitive data such as user data and so on. McAfee and Norton Antivirus are two instances of helpful programs that will protect us from potentially harmful interactions with malware or viruses that may put our customers' and company's sensitive data at risk.