

Gaming Room Software Document

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 | <mm/dd/yy> | Dae’Vante Twymon | <Brief description of changes in this revision> |

**Instructions**

## [Executive Summary](#_sbfa50wo7nsh)

An engine needs to be coded for a company named Gaming Room to develop an online web-based version of the game Draw It or Lose It. The game must support multiple teams, each consisting of multiple users. The users must have unique names, and their teams must have unique names. Additionally, the system must have a design constraint such that only one instance of the game exists in memory at a given time.

## Requirements

## [Design Constraints](#_2et92p0)

**Design constraint 1:** Uniqueness of Name and Entity: Each user and team have a unique name

**Implication**: a centralized database system can be used. In databases, we can use strict constraints such as not null or no duplicate to ensure each name is unique. Alternatively, to this, we can use validation blocks to check that the input name is not already assigned to any other player or team.

**Design constraint 2:** Only one game instance can exist in memory

**Implication**: the implementation of a singleton design pattern makes it so that there is a validation check to make sure that only a singular game instance is set to run.

**Design constraint 3:** Game design allows an increasing number of items and players while not degrading performance.

**Implication**: by using a design where you have a software design that will use the system server to run the game with a singleton design, so multiple instances of the game won’t impact performance.

**Design constraint 4:**The **application** must protect unauthorized access and user data.

**Implication**: The use of a database along with the use of multiple class instances stops the code from being easily accessible. The program driver is also disconnected from the main classes directly (under having an associated relationship to any of the instances) which requires multiple checks before getting to the main code.

## [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)

<Describe the UML class diagram provided below. Explain how the classes relate to each other. Identify any object-oriented programming principles that are demonstrated in the diagram and how they are used to fulfill the software requirements efficiently.>

**"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.**

**The UML diagram is made up of multiple entities containing class names in the top section, attributes in the middle, and maintaining methods in the bottom. The private member attributes have minus signs to signify that they are private while the methods tend to have plus signs for them being public though some constructors are private, so some classes have a singleton behavioral design.**

**Classes: GameServices, Game, Team, Entity, Program Driver, SingletonTester**

**Relationships: GameService -> Game a 0 to many Associations Relationship**

**Game -> Team 0 to many associations’ relationship**

**Game -> Entity inheritance relationship**

**Team ->Player 0 to many associations’ relationship**

**Team -> Entity Inheritance relationship**

**Player -> Entity Inheritance relationship**

**ProgamDriver -> Singleton Tester**

## [Evaluation](#_2o15spng8stw)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Operating System:  Built on UNIX which is known for stability and security  Hardware: generally made of quality hardware but comes at a high cost  Software: focuses on reliability and user-friendliness and has access to various tools and software for web development and server management. However, software size is still growing and isn’t as extensive as some other operating systems. However, Mac has much security built into the system and is generally at lower risk of malware.  It comes at an expensive price and has limited software options, based on the price of Macs and their complexity it can be more complicated and expensive compared to other options. It can also be harder to find people with macOS server experience compared to other options | Linux is open source and is free to use, distribute, and modify so there are dedicated communities for development.  Linux is also highly customizable and offers a lot of flexibility to make servers fit whatever server environment is needed for web-based applications.  The command line Interface is used actively  to  navigate through Linux giving access to more control and many automation capabilities. There are also many distributions such as Ubuntu, Debian, Arch Linux, Linux Mint and which can be used and have their strengths and weaknesses. Linux is cheap because of no licensing costs, especially since it’s open-source, Linux has strong security,  is stable and known for its stability, effective even for non-optimized hardware being able to run optimally on year-old systems,  Linux also can handle small and large deployments  It is good for scalability, and there is wide software and community support.  Linux is also good for databases being supported, has various deployment tools, programming languages, and web server support as well  The weaknesses, however, are that Linux has a huge learning curve, especially with the command-line interface, and there are compatibility issues with some apps and proprietary software. | Windows is a commercial operating system which means it has a lot of support for applications, especially being the most popular computer OS in the world. GUI is user-friendly and easy for various users to use, it has wide hardware compatibility, has ease of integration with Microsoft apps, and has an active directory for centralized user and system management.  It is the native platform of the .NET framework making it a robust environment for developing and hosting web apps, it has many development tools including visual studio for building web applications as well and there is extensive documentation, support, and community resources so help is readily available.  Cons however are the cost of licensing for Windows servers, Windows also isn't a UNIX system so it's more vulnerable to malware and security breaches which requires more diligent security practices. It is also more resource-intensive than something like Linux, which can impact performance significantly. Windows is also less flexible so it can be restrictive on server environments, and due to frequent updates and maintenance, it can require more reboots leading to inconsistent run times. | The benefit of having mobile devices run applications as web hosts is that they’re portable no matter what the OS is if you have a connection to the internet. The devices are dedicated to energy efficiency, they are also cost-efficient compared to other options including avoiding the costs of a dedicated server, plus e ability to leverage mobile development skills. Weaknesses include limited resources and the struggle to use resource-intensive apps or not having access to them. Connection issues are also more prevalent here because mobile networks are more prone to outages compared to competition, security issues because phones are more vulnerable to security breaches, mobile devices are unable to scale well, and depending on OS and OS updates it could disrupt the web functionalities, |
| **1Client Side** | For cost Considerations, Xcode, Apple’s IDE, is free; however, additional tools may need to be paid for. Macs tend to be more expensive compared to Linux, mobile, and Windows competition, and testing across multiple macOS versions and Mac hardware can mean more required training, making it more expensive. Included is also an Apple developer membership, which is another added fee.  There's time, UI/UX designs require extra effort in adhering to Apple’s human interface guidelines, and macOS optimization can involve extra development as well. MacOS expertise is also required, Knowledge of Objective C or swift programming languages are necessary, Xcode and interface builder also required, as well as skills in debugging and profiling. Not only responsive designs are needed, but thorough testing also has to be done and cross platform frameworks like Electron are needed to reduce the effort needed for the development effort. | Because Linux is more open source and uses a variety of programming languages, has many distributions it could take more time and money to learn especially because Linux has a significant learning curve since CLI (command line interface is a core component of Linux) which will increase time and money spent, depending on desktop environments used things like GNOME, KDE, and XFCE may be needed for supporting them, and because differing forms of Linux alone like Debian vs Arch Linux means that testing will need to be extensive as well. | Costs can vary since windows are widely used. It may be less expensive with training, especially since Windows has a focus on user-friendliness. It also should be the easiest of the choices for developing web apps, especially because most developmental tools can be accessed on Windows. Visual studio code can be cost-effective | For Android, there’s Android Studio, and for iPhone, there’s Xcode. Both are free, but the prevalence of plugins and libraries needed for cross-development tools can make them more expensive. Because there are hundreds if not thousands of versions of Androids and 15 different iPhones as well as varying device sizes and different ways, they can access the internet can be time-consuming. |
| **`** | For Mac, the programming languages are Swift (apple’s modern programming language), Objective-C (another Apple/mac exclusive programming language), and C/C++, which also have access to Python, JavaScript, and Ruby. The IDEs used are Xcode, AppCode, and Visual Studio code. | For Linux, the programming languages are C++, Python, Java, JavaScript, and Shell Scripting, and the IDE they have are Eclipse, Visual Studio code, JetBrains, and Vim/Emacs. | For Windows, the languages are C#, C++, Visual Basic .Net, JavaScript, Python, Java, and F#. The IDEs are Visual Studio Code and JetBrains Rider. | For mobile devices, the coding languages are Swift(iOS), Objective-C(iOS), Java(android), Kotlin(android), JavaScript(cross-platform), Dart, and C #.  The IDE needed would be Android SDK, IOS SDK, Cocoa pods, and Gradle/Maven. Flutter/react Native. |

## Recommendations

1. **Operating Platform**: The operating system I would recommend being used is Windows. Windows has the most versatility in terms of usable IDEs to work with. Windows is the most popular and widely used operating system and the team for the company would be able to easily adapt to Windows quickly which would also reduce the time of learning the learning curves of Linux and MacOS applications. And having mobile architecture would prove to be counterintuitive as it isn't reliable.
2. **Operating Systems Architectures**: Windows `1 is the latest version of Windows and it has increased the simplicity of the OS. The security measures also have gone up immensely. There are collaboration features built into the software, especially for software integration. Not only that but Windows has an easier and more understandable structure for data storage, and they have built-in purchasable cloud storage with OneDrive as well and it's easy to access off any device used in case of emergencies. Windows also has access to many database systems as well that allow the use of multi-tier architecture for extra protection.

1. **Storage Management**: For storage, Windows has Windows storage management that comes included with Windows 11 and it allows storage to be easier handled and allocated while also allowing for quick downsizing on other things taking up space on the Solid-state drives. There also is the inclusion of One Drive that can be used (as referenced earlier) that Windows has that can be scaled up to 2 terabytes worth of storage in inclusion to the SSD storage we have, and Windows makes it easy to integrate on the computer.
2. **Memory Management**: Windows 11 also improved memory management for memory allocation and loading being faster and more consistent. The processes were improved with Windows 11 to better utilize virtual memory space on the system which ultimately makes code, databases, and usage run more efficiently.
3. **Distributed Systems and Networks**: Windows offers Microsoft Azure, which offers developmental tools and application management built-in that helps with designing for usability on multiple machines for a network. Microsoft Azure is also a cloud computing server/that offers cloud computing services which would include servers and data being offloaded onto the cloud with a high availability architecture that has the goal of ensuring near 100% uptime on servers.
4. **Security**:

For security for the program, you would need to ensure that the code being used sets up safety measures such as the least privilege principle, especially online where trap doors and trojan horses are prevalent. This is especially important on UNIX systems like Linux and macOS (as well as things like Android operating systems / iPhones without built-in protections especially when the long search paths they contain are susceptible to Trojan Horse exploitation. Since it’s a web program and anyone can access it might be a good idea to use code that verifies system servers accessed have a recent AMD /Intel x86 chip or SPARC chip from Sun and have access to Solaris so that buffer overflows can’t happen or are less likely to happen.

With all this in consideration with Windows 11 and many of the recent Windows Computers that have been released (if we are not planning to build desktop computers from scratch), it might be a good idea to use high-end Windows PCs which typically come with the latest Intel chips. Windows also has access to most of the tools that Linux has for IDEs and security measures if not the same security measures. There’s also Windows security built into Windows systems and many Windows PCs that have come out for the past 15 years also come with McAfee for a free trial. Some have Mcafee built into the system that will warn of Hacks and trojan horses and spyware and malware attacks from files.