

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 | 07/13/2023 | Ryan Zackus | Creation of a cross platform game that is secure and uses methods to prevent server issues. |
| 1.1 | 07/29/2023 | Ryan Zackus | Updated evaluations for operating systems. |
| 1.2 | 08/11/2023 | Ryan Zackus | Updated recommendation section |

## [Executive Summary](#_sbfa50wo7nsh)

Creating a cross platform environment for The Gaming Room. The web-based game must have a way to check the memory for existing game Id`s and names. With unique names, the users will have the ability to check for existing games and if one is not found, a game can be created.

## Requirements

* Run on web browsers.
* Proper memory management
* Individualized game names

## [Design Constraints](#_2et92p0)

* Cross compatibility
  + Utilizing Java, the game should easily be able to serve multiple platforms.
* Users should be able to create and join teams.
* Unique game names must be used to help users get the right game.
  + Prevents naming conflicts and overlapping.
* The game should run on different web browsers.
  + Use Best Practices and extensive testing to make sure it works on all platforms.

## [System Architecture View](#_ilbxbyevv6b6)

## [Domain Model](#_8h2ehzxfam4o)

After Implementing the Entity class, Entity will hold variables from Game, Team, and Player classes. These three classes will inherit actions from the Entity and allow polymorphism by treating Game, Team, and Player as instances in of the Entity class. Name and id are private, ensuring they are protected in the Entity class. The Game class is a sub class of Entity, inheriting the id and name variables. The Game class holds methods to add teams and find a specific team. The team class is also a sub class of Entity, Team is used to add players and get the number of players. Player class contains information about one player that is in a team. Moving on to the GameService class. Singleton design patterns are used to make sure only one instance of a game is in memory at a time. This class takes care of the creation of games, teams, and players.

**"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)

Mac OS is known for security and staying up to date with their systems, but only Apple systems can use this OS which would raise the cost. Linux is open-source and can be very cost-effective while being stable but may need specific developers that are well versed to provide efficiency in development. Windows, however, is known for game support as well as tech support, which is necessary with the number of updates required to keep data secure. Windows is the most common OS and known by the majority of developers. This makes implementation quite easy.

As for the client side, we want the game to work on web browsers and mobile devices like smartphones and tablets. To flawlessly transition, we need to design the game to rapidly judge screen sizes and resolutions, and make sure it runs smoothly on popular web browsers. Optimization is required for the game to operate properly with keyboard, controller, and touch screen. To make sure the app runs perfectly, cross-platform frameworks and constant testing are essential. Developing for both web and mobile will require consistent open communication between the web and mobile development teams to make sure the user has seamless gaming experiences no matter what platform.

In my opinion Windows OS really shines through for this application. It`s popular, great for gaming, and would be cost effective, especially if PC`s are already in use. As long as the system is kept up to date, security should not be an issue. I believe, using Windows OS, The Gaming Room can have a cross-platform game that gives the games what they want, smooth, effortless interactions and less time worrying about if it will run and more time having fun.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac is known for being safe and secure. The system is constantly being updated and improved. But one is limited to Apple products, which are not cheap.   * $999 for Unlimited client. * MacOS supports server-Based Deployment. * LDAP - Supported * ADP -Supported * Cloud can be integrated with iCloud. | This OS is much more complicated than the others. Linux is open source, so there is a whole community working to improve security consistently. So, it is said to be very stable and secure. Many other tools and drivers are not compatible, it is wise to verify any tools currently in use are compatible.   * Roughly $2,500 for Datacenter server. * Linux supports server-Based Deployment. * LDAP – Supported * ADP – Not Supported * Cloud – Supported. | Windows is very well known by most developers. This OS has a plethora of server-side tech and has support for web development. Windows is also well known for its gaming capabilities. Security wise, Windows has more vulnerabilities than the other systems so more updates and maintenance are required.   * $ 6,155 for a Datacenter server. * Windows supports server-Based Deployment * LDAP – Supported * ADP – Not Supported * Cloud can be supported using multiple cloud platforms. | Mobile devices do not have the processing power to fully host a server. That being said, they can be used for monitoring the servers remotely. If cloud is being used, one would be able to access information with a mobile device.   * iOS and Android can connect to LDAP servers. * Only iOS has ADP support for accessing apps. * Both support cloud services. |
| **Client Side** | Macs generally hold a larger price tag than other PC options. Also, the tools specifically required for Mac development can increase budget. Depending on the Dev team, a major learning curve could cost time and money. | Cost wise, Linux can be a cheaper alternative. Being open source, it is free to use. With a range of compatibilities, one has more money saving choices. Again, if the dev team isn`t familiar with Linux then it can present a steep learning curve. | Windows is so widely known that users expect a certain user interface. Testing and getting it to operate properly can cost time and money. Experts in Windows seem to be easy to come by, comparatively. | Running a program for each OS on all mobile devises will require testing for each separate mobile OS, adding time and cost. Different languages are needed for either OS. Users expect clean and flowing applications on their mobile devices. |
| **Development Tools** | Swift is the main language for Mac OS. Xcode, being designed by Apple, would be the appropriate IDE for this OS. SwiftUI can also be used and is specifically for Apple.   * Xcode is $99 a year to upload. | C++, Java, and Python seem to be the best for Linux. Visual Studio as well as Eclipse are well known and would be a great IDE`s to use while creating a Linux program.   * Third party frameworks have licensing fees. | Once again C++ and Java are common languages and are used when making Windows applications. With that, Visual Studio and Eclipse would be a go to IDE`s for development. Windows also has Universal Windows Platform or UWP, a framework by Microsoft for developing apps. | Swift seems to be the most common for IOS as well as Objective-C. But for use in Android systems, Java would be the best language to utilize. Xcode for IOS and Android Studio would be the best selection for the appropriate OS. Using React Native, a cross-platform framework, one could use JavaScript and create apps that run on both systems.   * Android Studio has onetime fee of $25. |

## 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 ways Draw It or Lose It will be handled, Linux is the best fit. The resources are endless and continuously being updated by developers from all over the world. The money saved from utilizing all the open-source information could be put into developing the game further. Expanding the user availability and having the servers to handle even more players is easily achievable through Linux.
2. **Operating Systems Architectures**: For this application to be easily scalable, manageable, and store user’s data safely, a multi-tier architecture would be best. If the amount of players increases, using this method will prevent the game from becoming bogged down. The players have their own presentation level in the system, handling the user interface and display. The next level handles the parts that make the game run properly. Making sure players can talk to one another and properly play the game together. All while the database layer handles all of the data and game images, which can be accessed accordingly.
3. **Storage Management**: Using solid state drives will maximize the speed at which the game can pull image files and data from storage. SSD`s are also much more reliable than disk drives. Going over the top with memory can never hurt. With the growth of the application, it will likely be needed. Taking into account the multi-tier architecture, each system layer needs to be balanced to run at peak efficiency. One layer could bottleneck the entire system, slowing down everything. Utilizing Redundant Array of Independent Disks can protect data from drive failures. RAID 1 mirrors data, which means that the application would have a second copy of important information at the ready. On top of these factors, cloud storage can easily be utilized to scale and backup data.
4. **Memory Management**: Multi-tier architecture needs plenty of RAM to function properly. When choosing the RAM, go for dual-channel, high-speed RAM. Starting with 64 GB of RAM the game would run smoothly with 1000 players. If the player count continues to grow, then more RAM would be recommended to keep the game running correctly. With consistent monitoring of performance, one can easily see if more RAM is needed. Properly caching frequently used data will help performance, although, it should clear unnecessary files, or the system will bog down. Having the memory properly optimized and organized will allow the system to swiftly access what is needed.
5. **Distributed Systems and Networks**: The game would connect to a central server that handles the gameplay. The players can send invites to other players and join games through the server. This connection happens through the internet and the server keeps everything organized. It also makes sure that if one server has a problem, another server can take over, so the game doesn't stop. By integrating RESTful while using Linux, users can play together across all platforms, this allows the application to distribute the load across systems, making sure one system doesn’t get overloaded. These servers would process REST requests and send back appropriate data to the players after updating said data.
6. **Security**: To keep information safe, implementation of special security measures like passwords and encryption would be needed to protect sensitive user data. Using Linux is already a strong starting point when it comes to security. Linux uses authentication for access to certain data and it can easily support secure communication to keep the data safe as it moves around. All the security features on this OS are fully customizable, with new security features coming out consistently. Utilizing this OS to its full potential, one can create an impenetrable fortress as long as it is monitored and kept up to date.