

Draw It or Lose It

# **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 | 01/20/21 | Emily Nagorski | Identify constraints of project based upon the outlined software requirements. Outline the purpose and utility of the UML diagram. |

**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)

We are working with The Gaming Room in hopes of developing their web-based games known as Draw It or Lose it, for it to run on multiple platforms. Currently their game runs exclusively on android. When teams play the game, they will experience four 1-minute rounds, where they will try to guess the image slowly rendering on their device. If they are unable to guess correctly, competing teams will have 15-seconds to attempt to guess correctly before time expires. The Gaming Room is asking for the functionality to have this application supported by multiple platforms. Initial software requirements set by The Gaming Room include having one or more teams involved, each team should have the ability to have multiple assigned players, unique team and game names. Lastly, they would like there to only be one instance of the game stored in memory at any given time.

## [Design Constraints](#_2et92p0)

Design constraints set in place by The Gaming room are as follows:

* Draw It or Lose it should have the ability to run as a web-based game, that serves multiple platforms (game should run in a browser regardless of users’ device)
* Only one instance of the game should exist within the memory at any given time
* Need a web server to host the web application
* Storage method for all the images required for smooth game play
* Overall security of player data and information
* We need to stay within budget and on schedule

These requirements will need to be referenced and followed throughout the development process. This is just the beginning building blocks for the game; it is vital that the sign-in process for the game be seamless before we turn our focus towards game play development. The main hurdle for these design constraints is the functionality on multiple platforms. Because of this, we will need to consider having multiple teams working on creating the application for each platform, while having the game look the same on all of them, or we will need to inherit other languages into the existing code to help it run on other operating systems.

## [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)

When reviewing the UML diagram supplied below, we see GameService has a reference to Games, Games has a reference to Teams and Teams has a reference to Player. The Entity class was added to create an inherited relationship between Game, Team, and Player. Each of these classes takes and utilizes common information such as name and id, that are all stored within Entity. The UML diagram also shows that our design utilizes encapsulation frequently. We have many instance variables that are kept private within their own class. Team and Player have an inherited or a ‘has-a’ relationship because players of the game exist within a team. The UML also makes a distinction that the classes share a zero to many relationship. A zero to many relationship indicates that there may be zero of this relationship or several of this relationship associated with the given record. Each object in the class will either relate to many others, or no other objects. All these relationships will help ensure that the game runs smoothly and that multiple instances of games, players, and teams do not exist at one time.

**"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 must 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 the operating system put out and owned by Apple. It is a closed platform, so only Apple can release updates. Each time an update is released the web host must pay to get the latest version. Another cost associated with MacOS is that it requires Mac hardware to run the software. There is also a high cost associated with web hosting on Mac, and a wide range of prices depending on our specific needs. This does deter many companies from supporting Mac. With this price tag comes reliability and one of the most secure hosts. With this high level of security, it is likely that the data for our game would not get hacked. MacOS is Unix based and therefore some developers find the transition between Linux and MacOS to be quite easy. Along with similarities to Linux, Mac also allows flexible terminal commands for easy access, changes and creation. A downside like Linux is that some software would need a 3rd party, or virtualization system to be compatible, and this could increase the cost as well as security concerns. MacOS performance is optimized for Apache, which can provide access to libraries that can help make deploying faster and simpler. | Linux is often regarded as one of the most stable and reliable web-based environments. It is also free, open-source software, so there is no cost associated. They use open-source software, and with that provide multiple hosting options. Security of the application on Linux is also a large positive as it is generally much more secure than other operating systems. While Linux does have many positive attributes, it is often noted to be one of the more difficult operating systems to work with. Knowledge of using the command line, as well as being comfortable with manual edits of configuration files are characteristics of Linux that sometimes are off-putting to a newer developer. With Linux being based on the command line, you can easily interact with the server, and therefore configurations can be made without rebooting the system. The most important downside, in my opinion, is that Linux is incompatible with Windows applications. The only way to run Windows applications would be to use 3rd party software which would introduce further complications and security concerns. Therefore, if The Gaming Room requires Windows based applications on the server, we would not be able to host the game on Linux without the use of a 3rd party. | Windows is quickly becoming one of the most popular operating systems in the world. Because it is owned by Microsoft, Windows does include a licensing fee that would need to be added to consideration for The Gaming Room. Windows typically is more user friendly for development, and therefore more beginner friendly. Generally, more software is available and compatible with Windows compared to the other operating systems. Windows also supports third-party applications and is compatible with Microsoft applications. Windows offers automated updates, and therefore requires less administrative monitoring and maintenance. Overall Windows is more susceptible to malware, and other security-related errors, especially in comparison to Linux. | Mac, Linux, and Windows are all hosted and able to run on mobile browsers. Therefore, we would not need a server for any native applications. If the game is running smoothly on a browser, there should be no problem with having it optimized for browser support on a mobile device. If instead we want the game to run as an application for the mobile device, there would be additional costs and concerns associated with this. Considerations should include the cost of having an application on the App Store, approval by the App Store for store entry and any possible application earning fees. |
| **Client Side** | Developer cost and time should not differ too much between Mac, Linux, and Windows. The main consideration would be wanting to make sure that the developers working on the project have experience with the clients specific to each platform. Development time should be reasonable as, once the game exists on a web-browser, not much additional code should need to be written to optimize it for each browser. The main concern would be thorough and rigorous testing of the game on each browser. MacOS needs to consider Chrome, Safari and Firefox. | Linux will also not need a large amount of developing time, and therefore will not incur large costs to The Gaming Room. The biggest concern is just that thorough testing of the game on each client is completed to ensure smooth game play. The main clients that need to be supported by Linux include Edge, Firefox and Chrome. | Developers working on the Windows operating platform would need to have experience working with the clients most used on Windows systems. Again, thorough game play and testing will be their main concern after the initial development of the web-based version of the game. We would want the developers to make sure the game runs smoothly and identically regardless of the platform being used. Clients to take into consideration would include Edge, Chrome and Firefox. | Mobile device applications could require more specific expertise if we want the game to transition into a mobile app for iOS, as it already exists on android. If we only care about the app being accessible on a mobile device's web browser, the main hurdle would be front end optimization and appearance of the game on a smaller screen. That said if we want there to be an app for the game in the App Store on iOS, we will need specifically trained developers working on it, which could incur additional costs, and time. Getting an application on the App Store comes with a few more hurdles than just having a mobile browser game. Clients that would need support for a mobile browser include Chrome and Safari. |
| **Development Tools** | While there are many choices, popular frontend languages include JavaScript, HTML, and CSS. Backend languages that are relevant include Java and SQL. When it comes to IDEs, that is generally up to the developer preference and some IDEs do have an additional cost associated with them. A few examples of possible IDEs would include Eclipse, and Visual Studio Code. It would be useful to have a team responsible for the frontend team, and a separate team that is responsible for writing, and maintaining the backend. | Like Mac, frontend languages that could be used include JavaScript, HTML, and CSS. Relevant backend languages would include Java and SQL. As before, IDEs are all about personal choice. However, when working with a team, it is best everyone uses the same IDEs for production. A few IDEs that could be used would include Visual Studio Code, or Eclipse. My suggestion would again be to have a team who is responsible for the frontend, and a separate team in charge of the backend of the game. | The frontend languages again are HTML, CSS and JavaScript. The backend developers could use Java and SQL. Relevant IDES include Eclipse and Visual Studio Code, but of course are up to the preference of the development team. There are some IDEs that incur additional costs, but many free IDEs exist as well. As stated in the other categories, it would be my suggestion that there are two separate teams for the front-end work, and backend work. Browser updates are things to be considered, that could break our game, so even after the game is up and running there is still constant maintenance that will be needed to keep it running effectively. | Languages needed for creating an app for iOS would include Swift, Objective-C and HTML. Because of this iOS specific languages, you would need to choose an IDE like XCode, but could also utilize IDEs such as Visual Studio Code. XCode will make it easier to test your mobile app by running it on a virtual iOS machine. When it comes to mobile applications, I prefer to have one team handling the application, depending on the size of the game/audience. However, if the audience is large and the game is successful, having multiple teams involved would be a good idea. |

## 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**:
2. I am of the belief that Linux operating system will be the most appropriate platform to expand Draw It or Lose It. Linux is cost effective, reliable and can be run on any system architecture. This flexibility will allow for the game to run on multiple hardware systems, and provides maximum room for growth and expansion of our audience, all while having no additional licensing fees.
3. **Operating Systems Architectures**:
4. Linux is a UNIX based operating system. The operating system is made up of the kernel, system library, hardware, system and shell utility. The kernel acts as the core, and is responsible for the major actions of the OS and handles the underlying hardware. The system libraries can be referenced to specific functions, and are built in. System utility programs work to complete specialized activities. The hardware used in the Linux OS includes devices such as RAM, CPU and HDD. Lastly, the shell acts as an interface between the kernel and the user, it is responsible for taking commands and running functions accordingly. Working within Linux, it would be beneficial to implement a client-server pattern. This pattern considers our two main components, and will ensure clear communication between the two. The client will be responsible for initiating interactions and the server can react. The client and the server can both exist within the same system and I believe this is the most simple way to navigate our game.
5. **Storage Management**:
6. For responsible and logical storage management it will be necessary to consider how much space is needed for the game as well as potential growth. It would be most logical to make use of the network files systems (NFS) already available within Linux. This system makes it possible to transfer files between computers running window and UNIX. We will also want to create a partitioned repository so that each partition is no larger or smaller than needed, thus ensuring our memory is being put to good use.
7. **Memory Management**:
8. Paging is a method commonly utilized withing Linux OS already and would work quite well for this project as well. Paging allows for linear memory; therefore, memory addresses are mapped to pages and not to physical memory. This method allows for easy swapping in and out of RAM and therefore leaves more memory available for use than is physically available. Only pages which are in active use by the program need to be mapped into physical memory. This method can allocate more memory than what is physically available, and therefore would be an excellent use within our game, as not all images are needed at the exact same time.
9. **Distributed Systems and Networks**:
10. Linux operating systems can utilize containerization. Containerization is the packaging of related code, libraries and dependencies to create an easily executable task that can run consistently on any infrastructure. This will allow for faster and more secure deployment. With the method of containerization, Kubernetes can also be utilized. Kubernetes is an open-source platform that manages containerized workloads. Essentially, they create server nodes that applications can run on, and using these nodes can assist in the alleviation of system outages. In theory, if we were to have multiple clusters of containers servicing our customers and one cluster fails, it may boot a user from the game, but overall, the effects of the data failure would not be detrimental to the game.
11. **Security**:
12. Linux is regarded as a highly secure OS. Linux has implemented a user privilege model, which offers many built in security defenses to aid in protecting against attacks. Linux has a leg up on Windows, generally in Windows OS everyone has some degree of admin access. Linux restricts root access through the user privilege model, only allowing your typical user enough access to complete ordinary tasks. This lack of automatic access rights creates an inherent safety of our game data. Linux also has built in firewalls within the kernel, administrators have the power to add additional security to the kernel as well if desired. This role-based security plan will ensure that our users have the access they need for smooth game play but restrict their actions enough so that game and user data remain safe.