

Draw it or lose it

# **CS 230 Project Software Design Template**

Version 1.2

## 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 | 09/14/2022 | Hatcher Blair | V 1.0 |
| 1.1 | 09/28/2022 | Hatcher Blair | Updating previous comments to reflect new requirements |
| 1.2 | 10/12/2022 | Hatcher Blair | Updating the recommendations section |

**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)

This software will run the game draw it or lose it. It will handle the creation of different teams, assigning players to teams, the creation of unique team names, and will allow only one instance of the game to exist at one given time.

The game is now ready to be scaled to several different platforms and potentially have thousands of users, what are the necessary considerations to scale the application up to multiple platforms?

## [Design Constraints](#_2et92p0)

<Identify the design constraints for developing the game application in a web-based distributed environment and explain the implications of the design constraints on application development.>

* The game must only have one instance running at a time
* Needs to allow for multiple teams with multiple players, and they must all be unique
* Must support multiple different browsers and hardware configurations to allow maximum compatibility
* Must run on mobile devices

## [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)

Most of the games services are based around the entity class. This allows for easy changes later in the development cycle if needed. Game, team, and player all extend the entity class meaning that all of these classes have access to all the features of an entity. The program is also structured to only allow once instance of the game at a time, or a singleton pattern. All the different classes are structured to not allow outside access to critical information providing enhanced security.

**"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 really exist in the server space. Especially with their new arm architecture. There is limited or no repairability and not much customization options making it a poor choice for commercial clients.  Does not offer a server-based deployment method that will allow for maximum compatibility because modern Mac OS runs arm. | Linux is the best OS for hosting servers. With its multiple distros, customizable feature sets and the ability to run on several different types of hardware it is an excellent choice.  Offers a server-based deployment method. Licensing costs will vary depending on how we deploy the server. | Windows is ok for hosting a server. It can be done, but you are not able to have the same level of control as you can with a Linux system. Windows is also a heavy operating system meaning you are giving up performance to the OS that could be used for processing user requests.  Offers a server-based deployment method, licensing costs will be very expensive. Each machine will need a licensed copy of windows. | You don’t want to host a server on a mobile device. There is not enough processing power for serious use cases, and you run into the same issue as you do with mac OS, limited customization and repairability.  Does not offer a server-based deployment method. |
| **Client Side** | This is a web-based application, so we will handle the processing on our servers. This also means that we need to optimize the website to be able to run on safari. If the website is set up to adjust for different screen sizes, then we are good to go.  Very little cost to get the website working on Mac OS. We just need to verify that our API’s function on ARM. | Linux can have some compatibility issues. We need to make sure the website is optimized to run on various Linux distros.  Potentially a lot of headache and cost getting the application to work on Linux. There are a lot of different distros that users could be using. Ultimately Linux is a very small portion of users and so Linux support should be something added later, not in initial release because of all the different variables. | I don’t expect to run into many issues using windows, we just need to make sure it will run well on edge, even though a lot of windows users use chrome as their browser.  Windows will be very easy to run on, there shouldn’t be any compatibility issues and will cost very little to implement. | For mobile development there is more to consider. We need to make sure the game is compatible with touch inputs and easy for mobile users to interact with the same way a user with a mouse and keyboard would be able to.  Could be very expensive to optimize for all the different android OS’. Safari an IOS can also have some strange interactions, but because all iPhones run the same OS we only need to optimize once. IOS also has more users than android and so we will start with IOS and the most popular android OS’ and then we can move onto the less common android OS' if time and budget permit. |
| **Development Tools** | We are going to use Java for the base application, HTML to code the website and run the java on the user side, and we are going to need a graphical design software to design the game.  There will be no additional development tools as it is a web-based application, we can use our existing tools to optimize for the platform. | We are going to use Java for the base application, HTML to code the website and run the java on the user side, and we are going to need a graphical design software to design the game.  There will be no additional development tools as it is a web-based application, we can use our existing tools to optimize for the platform. | We are going to use Java for the base application, HTML to code the website and run the java on the user side, and we are going to need a graphical design software to design the game.  There will be no additional development tools as it is a web-based application, we can use our existing tools to optimize for the platform. | For mobile we need some special considerations for screen size and touch input. If we account for mobile in our HTML it will work.  There will be no additional development tools as long as we decide to keep it as a web based application for mobile. Depending on the hurdles we encounter it might be better to develop applications for IOS and android that we can put on their respective app stores. This will require a developer apple ID account, which has a nominal cost, as well as an application to develop code for IOS, which is free. |

## 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**: The server should run on a linux based server for its feature set in the server space and to allow for easy expansion down the line.

I recommend the storage servers run unraid for its ease of use and easy expandability later down the line. In today’s world there are solutions that will offer better performance, but we don’t need a very performant storage server and so a basic server running unraid will meet our needs. We can configure the servers in a RAID 10 configuration to provide a good balance of read and write speeds as well as data redundancy. In this configuration we can lose one drive in each set of two without losing any of our data.

For our machines hosting the games I would recommend using ubuntu. Ubuntu has a lot of benefits, but the primary one is the reliability and the widespread use in the linux community. Most people that use linux have used ubuntu and so there is a lot of familiarity with it. It is also a very feature packed OS, which does make it heavier than other options, but it means that we don’t have to worry about not having a certain feature that we need and needing to migrate OS’s down the line.

1. **Operating Systems Architectures**: Linux will run on an x86 or an Arm architecture. We will be using x86 for this to make our code more compatible with different server configurations around the world.

The machines that we are deploying will be using the x86 architecture. This is what almost all modern PC’s run. x86 has an expansive instruction set due to is complex instruction set computing (CISC) design. This means that there is an instruction for almost any task we would want to do. This also allows for maximum compatibility with the server components and configuration as almost everything is x86.

1. **Storage Management**: We can store any information that we need to on our servers. The program can write and read from files on the server or on our other servers around the world. If we need more storage, we can buy another server or upgrade our configuration later down the line.

I recommend the storage servers run unraid for its ease of use and easy expandability later down the line. In today’s world there are solutions that will offer better performance, but we don’t need a very performant storage server and so a basic server running unraid will meet our needs. We can configure the servers in a RAID 10 configuration to provide a good balance of read and write speeds as well as data redundancy. In this configuration we can lose one drive in each set of two without losing any of our data.

1. **Memory Management**: Linux will dynamically allocate memory to the game depending on how much the app is requesting and what other services are running, once an instance of the game is closed all that memory is freed up.

Linux and therefore Ubuntu has a very complex memory management system. It will manage the system memory very well on its own, but we can also tune a lot of the memory management settings to fit our needs. This means that we can tune our memory to function how we want it to when we want it to. For our storage servers we will set aside some of our memory as a cache so that when data is being written or read from the server, we can dump it all in memory and let the slower read or write happen in the background freeing up the network for other requests. As for the game servers we will not need any tweaks to the memory management system if we design some management into our application. We will need to make sure that the program frees up memory when it is done using that space and requests more memory when it is going to need a little extra for its next task.

1. **Distributed Systems and Networks**: The game will be hosted in several different data centers around the world and so if one data center has an outage there is another one for users to connect to. This allows for users from all over the world to have better connectivity than once centralized server.

To have the best reliability, uptime, and global coverage for our game I would recommend using amazon web services (AWS) to host all our servers. They are one of the market leaders in the space and have a vast array of products to chose from, including servers that are specifically designed for web and mobile applications. They also have plans that are for storage servers and so it makes a lot of sense for us to outsource the server hosting to someone who has a lot of experience and customizability for this application. AWS also has datacenters all over the world and almost never has any downtime and so we can expect a reliable network to host our game on that will cover the entire world.

1. **Security**: All of the users’ data will be stored on our encrypted servers, except for any sort of payment processing that might be required, payments will be handled by a third party. This allows us to encrypt all the user data on our servers, and there will be redundancy set up and so if once server goes down then we can retrieve their data from another location. This also allows the user to access their account from any device.

AWS datacenters are extremely secure. It is impossible to get physical access to any of their locations without the proper clearance. Even as a customer of them we will not have physical access to the servers. This means that the likelihood of a physical attack on our network is slim to none. There is also a suite of features that are offered for preventing and detecting virtual attacks on our network. A lot of these tasks will be automated, such as data redundancy on our storage network, and so we get to spend less time worrying about security and more time worrying about the future of our application.