

<The-Gaming-Room >

# **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 | <05/21/2023> | <Andre-Burton> | <This is the initial setup of “The Gaming Room”, added changes to the cover page, document revision history table, executive summary, requirements, design constraints, domain model, evaluation, and recommendations to progress current model into version 1.1> |

## [Executive Summary](#_sbfa50wo7nsh)

The summary of the game Draw It or Lose it is developing a game which produces an image on a screen that helps the player guess the puzzle. How this is achieved is there will be four images shown within a 30 second interval if the guess is not correct then another team has an additional 15 seconds to guess and solve the puzzle, the game has four total rounds.

## Requirements

* Web based.
* Have options for one or more teams to play.
* Can assign multiple players to each team.
* Custom team names which can verify are already used so there aren’t duplicates.
* Prevent multiple instances of the game in local memory at a given time.

## [Design Constraints](#_2et92p0)

With the requirements laid out there displays a few design constraints, one being the amount of time needed to take the existing android-based code and developing for a web-based model, next will be developing it so different web-based programs can use it correctly (Chrome, edge, Firefox, safari, etc.). How to take clients web information to prevent overuse of storage which may result in a slow and unusable program. Must implement coding to remind user if more than one instance of game is already in use so there is no confusion as to why it may not be working. Somehow create a way to prevent users from replicating usernames.

## [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 looking at the below UML class we can get a better understanding of what is happening in the game by breaking it down into classes and how each class not only plays a role in the game as a whole but how it directly influences different classes. When looking at the diagram we can see the entity class being the main class which everything flows into meaning that all information must go through it. Below the entity class we have the team class which takes team specific information such as (names, add player information, and then can put them into lists) This information can then transfer and read information from its surrounding classes to better set the game up. The class to the right of it is player class which is where player names can be chosen and then the information is funneled into the team class to initiate teams from the players. The Game class then takes the player and team information and adds the teams to the game and starts to create the games information. Next, we have the GameService class which gets a lot of the game information and then adds it with a lot of getter and setter code. All this information is then set into place and the entity Class ensures proper placement and usage. Each one plays a direct role in ensuring the game is set up properly. When looking at OOP principles we see abstraction with private and public portion so the code which helps to hide unnecessary information to the user. We can also see the OOP principle of Inheritance in the form of taking features from one class and sharing among different classes which helps to create cleaner code. These are examples of a few OOP principles in play with there being many more OOP principles present in the UML Diagram.

**"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** | Pros:  - Much more user-friendly interface  - Less bloatware  -Better security options  - Integration among other apple devices including mobile apple devises  - More consistent updates which makes changes to app easier  Cons:  - More expensive than other software options  - Hardware changes are almost impossible  - One of the smaller user groups compared to other options  -Xcode is not free and will cost $99 dollars a year to use  - will need to be able to accept lots of different models sporting different levels of performance. | Pros:  - Open Source making it accessible to anyone  - Extreme security options preventing possible viruses and insecurities for users  - Faster than other options due to its simplicity  - Low system requirements meaning older devices can still run it  -Most IDE options are free  Cons:  - Learning curve compared to other languages  - Software issues as many programs are not capable of running on Linux  - Drivers issues as these drivers don’t exists with the Linux system  -Will need to work with older devices who may be running older devices which may require more needing to be store on the server | Pros:  - Largest group of clients out of all OS.  - Super easy transition from older to new software upgrades  - Tons of driver support  - Tons of supported language options  -Most IDE options are free  Cons:  - Has highest amounts of Virus opportunities  - More expensive than Linus  - May be more complicating to use  - Not open source making it difficult to make some changes  - Very Ram usage heavy  -will need to accommodate to large amounts of users accessing the server | Pros:  - Good built-in security  - Easy user interface and back up options  - very cost friendly with tons of price ranges  - easier application management among users  - simple and easy design  -Java IDE are free  Cons:  - More than one mobile operating system exists  - quicker hardware updates than other OS  - Lots of privacy options  - Simple user interface due to less user hardware options  - No upgraded hardware options  -Xcode cost $99 dollars a year to use  - Will need to account most devices being lower powered meaning the server will need to take on more of the workload. | |
| **Client Side** | Mac is user friendly due to its simplistic OS, works with any apple product for both mobile and mac-based devices, due to its specific language an expert in the field will be needed, Cost will be higher as well as most products cost more money than there Linux or windows options. Lastly the time level will be lower due to one product working for most of the OS lineup. Safari is the main browser for MacOS users which makes it simpler for the development team as it means they only need to text for one browser to verify if connects properly with the server side. They also will need to take into account how much they will want on the server-side vs the client side as the power difference between macOS users’ difference greatly due to the vast difference in performance among the lineup and generations of Mac systems. | Linux will be the cheapest option as its open source and older and lower cost hardware is able to run it, it will be the lowest time option as has many language options and there won’t need to be specific expertise needed to code in Linux due to its openness, the downside is there is a significant lower amount of end users using this OS. Linux comes with lots of browser options which means the development team during the testing phase will need to test whether the program works for different web browsers. Linux will also usually be used on older devices that have smaller storage options and slower hardware which in turn will require more of the information needing to be stored on the server side or the application will need to be made custom to account for on average a slower system. | Windows will cost a little more than Linux but will be cheaper than Mac due to the amount of cheaper hardware options, Time will be average as there are many more experts in the language of C++ available. This OS also has the largest clientele base making it the most profitable once running. Windows is much more open which means there could be multiple browser applications which could be used to access the application on the server side which just means that in the testing phase the development team would need to test to make sure each browser can access the information on the server side. With computers usually being faster and having more storage the information can be better shared among the server and client which will result in needed less internet bandwidth and a faster experience for the user. | | Mobile operating system will cost more than the other options due to needing code for more than one OS this will in turn increase time and expertise needed as there will need to be two sets of code written which will account for both IOS and Android. The upside is This also has a huge user amount which should in turn make more opportunities for application usage. In order for the client to access the web-based software each one will need to have a dedicated search Brower usually safari for IOS and google Chrome for android which comes pre-installed on the devices so actual implementation for the client side is easy and requires only know the proper search to connect to the server. With mobile devices usually having smaller memory capacity this usually will result in most the information needing to be stored on the server side to prevent overload to the mobile OS which will result in mobile users needing better connections to the internet. |
| **Development Tools** | Mac uses Object-C for its coding language which uses Xcode as apples main IDE and the only one apple uses. This may impact the team as this requires someone who has dedicated knowledge with this IDE rather than other OS who may be open to multiple IDE and the developer can choose which IDE which suits them the best. Xcode also comes with a price of $99 dollars a year whereas other IDE may be free. | Linux primary uses C as its writing language but can also support its language being written in Python, Java, Perl, and a few other smaller languages, Linux also has many Ide options such as visual studio, eclipse, and even sublime Text, which is only three of dozens of IDE that are usable. Due to multiple options of IDE their chance of a developer who is knowledge in one is very high needing more time to learn a specific IDE is not needed. These IDE also have free options so there isn’t any cost designated in IDE usage. | Windows uses C++ as its primary language to create apps and markets visual studio as its go to IDE but another ide which supports C++ can also be used. Due to windows using C++ this also leaves a good amount of IDE open to being used which then results in lower times as developers can choose which IDE suits them best that works with C++. There are also no dedicated costs when it comes to the IDE usage which will lower costs to the company. | For mobile devices we can look at android and IOS as those are used on most personal mobile devices, IOS same as mac uses Object-C as its language and thus uses Xcode as its IDE. Android on the other hand uses java as its primary which has dozens of IDE available to it with Eclipse being the main IDE. The cost of this will depend on which mobile system you’re using IOS will use Xcode which will come with the $99 dollars a year cost and will also require a person who has experience in using this IDE. While on the other hand android using java comes with multiple IDE available which in turn gives developers options to choose form most which are free, which lowers the cost and the required experience. | |

## 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**:

Due to the overall pros and cons of the different OS, windows would be the recommended Platform due to it having the largest customer base, easier to work with languages which require less expertise, this in turn will make the product easier to develop, cost less, faster to push out, and making updates and changes will be easier due to windows OS having easier to transition updates. Windows also has tons of different version which means we can make small edits to make the game accessible to all versions of windows creating a larger customer base. The other platforms could be expanded to down the road but weren’t recommended for a few reasons. Apple wouldn’t be recommended because the customer base is smaller, we would need a group of people to be experts in Xcode which is specific to apple which would increase time and budget. Linux wouldn’t be recommended either due to its significantly smaller user group who would use this game. Lastly, Mobile devices would be a close competitor but due to needing to code to multiple platforms such as android and IOS it creates the same issues as Mac with the increased time and budget to make it accessible to all mobile users.

1. **Operating Systems Architectures:**

Windows provides an OS that uses languages and IDE that are widely accepted and practiced. Its language is also more acceptable to people as it has multiple language options such as windows (7,8,10,11, etc.) which can be coded for the users and then update easily. Windows also has a system mode which gives the user unrestricted access to the systems software and hardware which is very important to application building. The requirements we would need to build properly in windows environment would be a team of developers proficient in C++ and the appropriate server/client connection models. Windows architecture is built up of both API (Application programming interface) and GUI (Graphical User interface) each one playing a role in the game’s development. The API will include the base code, Virtual address space, system resources, Process ID, execution thread, and security content which each play a specific role in the games development and running. The GUI on other hand consists of system resources, audio, visual effects, and many more options. Both the GUI and API help build the windows architecture which will play an important role in the games development and its usage once pushed out to the end user.

1. **Storage Management**:

Windows OS comes with its own built-in storage management system known as Disk management which comes with tis own benefits such as easier file transfers and easier data allocation and usage information. This system makes it easier to understand where the storage is going and how it is being used while having built in processes to help error check and defragment portions of storage. Disk management also comes with lots of built in tools which can be implemented to help with management of the game, these tools include new drive set-up, extend volume, shrink volumes, and new driver letter options. It also comes with free up disk space options which looks for data that is old and clears up waste, defragment drives which helps to reorganize data files making it easier for the drive to find information. All these tools are already built into windows meaning we can piggyback from those rather than create separate code to manage the game making the games development quicker and more budget friendly.

1. **Memory Management**:

Windows OS also comes with its own built-in resource management which includes resources such as CPU, GPU, and memory usage which will come specifically handy in application usage. We can use the Windows OS resource monitor to monitor memory usage in our application and then help to close unused data based on time and usage amount and build in a run and kill code to allocate a specific amount of memory for this designated application. Memory management also has dedicated function which will come specifically useful for our game and should be built around it these functions include reserving and committing memory, guard pages, enumerating’s, process heaps, file mapping, AWE’s, and memory allocations. All these tools will come especially helpful in creation of this game and due to these resources being built into windows it will assist in creation of the game resulting is quicker build times and overall lower budget needed.

1. **Distributed Systems and Networks:**

To build this application so it can communicate across multiple different platforms we need to create a web-based code that takes inputs form the user as integers and strings then outputs that information into the specific language for the device that is being used this makes it so that multiple languages can use this while at the same time having it communicate properly among different OS languages. How this can be implemented with windows is using their own distributed systems via servers which users will connect to using their search webs. Using the computers information we can then have the game allocate the amount of storage and memory needed between the server and the computer to have the game run properly with older devices we may need for support from the server rather than on newer and more higher powered system which can take more work load onto itself helping to free up server work load, this thus creating optimal performance for all clients on various platforms, this will help will connectivity, and lower the chances for outages and server crashes.

1. **Security**:

Windows has a built-in security program called windows security which we can implement into the app to watch and check for viruses and other security situations that may arise. Windows also uses this same application for its online application called windows edge which means that we only need to develop an app that works with windows security and don’t have to create its own built-in security from viruses. We would need to create some sort of personnel security into the app for privacy and user purposes which can be created using username and password options which then can be backed up by two factor authentication which is a standard in personnel information security especially in online databases. This will protect the user on there own personal level but windows also has built in server protection which consists of server hardening guides which will help to better protect the server side of windows systems. Some of the tools include secured-core server which stores the crypto key on the servers CPU helping to make it more difficult to access. Firmware protection also helps to scan for firmware updates and can also help with memory access protection. Lastly, it also has virtualization-based security which stores user credentials in a secure folder that the OS on user computers can’t access which protects users’ information in case of a malware attack. These are just a few of the many forms of security built into the user’s machine and the server’s machine which protects the user and servers’ information.