

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 | 1/28/2023 | Max Harris | Building out the team and player classes |

**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)

The program currently does not allow for the instantiation of multiple games, teams, and players. We will be adding the functionality to do so, storing the ID for each on a list to be obtained later to ensure that there are never multiple copies of the same ID.

## Requirements

*-*able to have one or more team per game

-each team has multiple players assigned to it

-game and team names must be unique to prevent duplicates

-only one instance of the game can exist at a time

## [Design Constraints](#_2et92p0)

-There will have to be a limit to the number of teams, or the UI and the pace of the game will get clunky and hard to manage.

-Once again, there will need to be a limit, and for most games, there will need to be some means of ensuring that the teams have an equal number of players.

-There will need to be some sort of documentation regarding the way that the unique IDs are created in case any manual testing needs to be done after completion.

## [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)

The ProgramDriver will instantiate the SingletonTester. The game, team, and player classes are all children of the Entity parent class. This is an example of inheritance, and it’s nice for the developer in this context because all of the children classes need to have an ID and name, which the Entity parent class has and passes down to them.

**"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** | It is capable of running Windows and Linux applications in tandem with Mac apps. It is one of the more expensive options, especially when it comes to long term maintenance. Security is pretty good. The hardware--though expensive—is well made and optimal for the OS. Hardware options are also more limited than Windows and Linux. | You can run Linux on so many machines, and it’s very easy to make it cater to your specific needs. It’s also very affordable, and there is a pretty large community surrounding application development for Linux, which makes long-term maintenance easy. (it’s also affordable, because the litany of hardware options continue to assist here). Linux also has the capacity to be highly secure, and there isn’t as much malware targeting it as Windows and Mac. | Windows is one of the less secure options, but it’s an affordable option with lots of options for hardware. Generational inconsistencies negatively effect long-term maintenance. | It can be difficult to expect what sort of compression might happen to different types of files when working with mobile. Hardware requirements can be difficult to meet, and if you want something highly portable (capable of working the same on multiple systems) then you’ll likely end up spending a lot of money. |
| **Client Side** | Not many games are developed for Mac, and not many people use Macs to game. They are ideal for anything graphically heavy, such as video editing software. Long-term maintenance can be more difficult, though, since it isn’t as common as Windows. (less people who know how to work on them) This also means it can get expensive. | While hardware options are near limitless, software options are near zero. Most big applications are not developed for Linux. Linux is affordable, and it’s ubiquitous in today’s gaming world. | Windows is very inexpensive. It is also very commonly used, which means that developers are familiar with it, and so are users. Generational inconsistencies can be difficult to manage when it comes to long-term maintenance, though. | Very convenient for the user. Depending on the OS and the hardware requirements, this can be a very inexpensive route for the user as well. It will however be very expensive to get your mobile app out to the public, and to maintain it long-term. |
| **Development Tools** | Very well connected to iPhone and iPad. SDKs promote integrated development for each of those platforms. Once again, there isn’t as large of a community for game development on Mac, so there is less support and fewer tools available. | There isn’t much support for Linux across some of todays useful development tools. There is sometimes issues with file formats when transferring from Linux to other OSes. It’s going to be up to you to get Linux on your machine, also. | Very popular amongst gamers. There are tons of different libraries provided by Microsoft for writing windows applications. There’s also the UWP framework which is super convenient! Especially since it makes it easy to port games to Xbox systems. | It can be difficult to develop for mobile since testing your software on the actual device can be a rigorous process. It can also be difficult to expect the little differences between all of the machines that might end up running your software. |

## 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**: Given this context, my recommendation would be to use a physical server running either Windows or Linux.
2. **Operating Systems Architectures**: Basically, you’ll have an secure, isolated computer running the server side of the game. It will be there, ready to send/receive messages from each client so that users can play together remotely. Instead of using information from within their own machine to update their game, each client will update the game using information that was sent to the server by other clients. The server just needs to have a way of securely managing that information as it comes.
3. **Storage Management**: For this product, the images should be stored using an online database. Each client can access the database, grab a number of images upon starting the program, then convert them all to binary data and store them as buffers. This way, the images can be updated and changed by simply accessing and editing the online database.
4. **Memory Management**: Each client will use their own memory to make updates to the game. The client will access the server to get vital information about each “guest” player. That information will be stored in the appropriate parameters in each client’s respective “guest” player in each of the other clients’ games. As far as how that should be managed locally, each client should only access the assets they need to access at any given time. This means that each game should only select one image from the buffer of binary data representing the images selected from the online database. That information can either be converted to an image and temporarily stored until the end of the round, or it could be converted to an image each iteration. The former is likely more optimal in almost every scenario, so it is my recommendation.
5. **Distributed Systems and Networks**: To deploy this game on multiple platforms, the main concern is with graphics. You’ll need to programmatically determine the platform and then access the GPU accordingly. If one server is down and needs maintenance, then write a failsafe into the matchmaking code that excludes players from any service that is marked as needing maintenance. For player-arbitrated matchmaking, send an error message to the player explaining why this is happening and when it will be done.
6. **Security**: User data should be encrypted at each end of the communication process. This means that when a message is sent from one computer to the server, it should first be encrypted. Then, when the message is received by the server, it should be encrypted using a different process, then sent to the other clients thusly. When communicating with the database, the information should be encrypted in the same way, and the database should store the data in that form.