

THE GAMING ROOM

# **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)5

[**Recommendations**](#_m8aleynsvzvc)6

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 02/20/21 | Christian Drouin |  |

## [Executive Summary](#_sbfa50wo7nsh)

The software design requires that there can be multiple teams, and multiple players allowed on each of those teams. This should be easy to solve by creating a class for teams with a subclass involving players. Within the team class we can ensure that only unique names are used to meet the requirement of checking whether a name is in use. The design also requires that only one instance of the game can exist in memory at any given time. Using a singleton patterns, we can meet this design request in a highly effective way. The client requires but does not currently have a super class for getting and setting id and name. All subclasses of Game, Team, and Player will inherit from the Entity class that we will create so we can eliminate repetitive coding. As it stands currently the client has getters and setters for each subclass that are all doing the same thing. Creating this entity class will improve the simplicity of the clients’ code while maintaining industry best practice.

## [Design Constraints](#_2et92p0)

One of the major constraints of this project could be the client’s hardware. We don’t know exactly how many games, teams, and players the client has in mind in relation to our software. If they do not have the proper hardware, or server software the application could become overloaded and crash. There is little security involved in the design of the client’s software. Being that the design does not require a formal authenticator the client could run into some serious malicious threats to their application.

## [Domain Model](#_8h2ehzxfam4o)

This UML class diagram is designed in a way to be highly efficient. The base class labeled Entity is used to hold methods and attributes. The attributes for Entity are id and name. These attributes are an example of object-oriented programing because they are allowing us to assign information to help us describe an object within Entity.

Falling below Entity and thereby inheriting all methods are Game, Team, and Player. Although all three falls below Entity let us first explain GameService. GameService is a class used as a singleton service for the game engine. Essentially GameService is ensuing that there is a list of games and that there are not duplicate games in that list. GameService is there to keep track of all items created and is created using the singleton pattern so that multiple lists of games and information are not created. All classes chain from GameService to ensure that all information is stored safely. This class can not be altered from other classes.

The next class Game is created to allow for the instantiation of multiple games. Each Game added is an object to GameService and Entity both. With a Game object created multiple Teams can be created and then players can be added to those teams. The classes Game, Team, and Player are examples of object-oriented programing. Game creations are objects to GameService, then Team is a further object breakdown of Game, and then Player is a final Team object breakdown. Each has its own set of attributes and methods that help manipulate and explain its objects. These classes are used to meet the requirements by allowing multiple teams to be created within a game since it is a subclass. This is continued down to multiple players allowed on a team as well.

****

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac offers extremely secure software. Runs very smoothly. Tends to be very expensive | The cost of a Linux based system in considerably lower than its competitors, but comes with a low level of security | Windows can run many applications its competitors cannot. It can also be very slow and filled with bugs. | Mobile can be accessed anywhere by any device allowing for greater traffic. Can be slow to load, and very expensive. You will need to use the cloud. |
| **Client Side** | There will be an exceptionally high cost to maintain a mac-based application for many clients. Mac tends to be the most expensive. Due to the extensive security measures this could increase time to produce as well. | This would be a great choice cost wise for a high number of clients. Linux would be quick to work with also. Because of its low security this choice could put the high number of clients at risk of an information leak. | Windows can meet in the middle. This choice won’t be too costly and will be easy to work with. The many clients could bog down the system for the clients. | This is a great choice as far as a client is concerned because of the ability to be accessed anywhere. This could be a considerable amount of money to keep up with and being a gaming application, this system could be quite slow. |
| **Development Tools** | The main programming language used for building apps with Mac is Swift. Some Python is used, but mostly on the back-end development. We will need to work with Swift to deploy the app through Mac | Linux doesn’t have a specific language to use in its app development, but C++ and Python are known to be an excellent pair for Linux development. You can really use any language choice. | There is no specific language dedicated to Windows app development. Windows does have its own IDE called Visual Studio which may be beneficial to use in the development of a Windows app! | Android and mac are typically built using their own respective programming languages. Kotlin and Swift. Because of this you are going to need some type of cross-platform framework such as maven or nuget. |

## [Evaluation](#_8h2ehzxfam4o)

## Recommendations

1. **Operating Platform**: After careful consideration I have decided that a Linux operating server is the best option for The Gaming Room to expand Draw It or Lose It. This choice wasn’t obvious and there were certainly other options that weighed closely. My two major choices were Windows and Linux. Mac was far to expensive for the client and another option of using mobile devices was out of the questions because they are notorious for bogging down with heavy demands such as a gaming application. At the end Linux took the job because of its cost and speed in comparison to Windows.
2. **Operating Systems Architectures**: Linux has the following five major components that make up its architecture: The Kernel, Hardware layer, System library, Shell, System utility. The Kernel holds all the background information that is required for the system to work. The System library is a collection of programs maintained and managed by the operating system. The System utility provides the user most of the tools of Linux. The Shell is the interface that allows a user to access and use Linux.
3. **Storage Management**: The Gaming Room should use a traditional cloud-based storage management system. We are not saving critical information from our clients and none of the data used in this application is going to be sensitive or large enough to require any type of specific specialized storage management. We are mostly saving email/password credentials for logging in.
4. **Memory Management**: Linux is known for creating a virtual memory layer. It then takes all the requests and assigns them to virtual memory which is a mixture of RAM and swap space. This extra step involving a virtual memory layer ensures that each request doesn’t try and use memory already being used by another request. Linux is known for being one of the fastest operating systems in the world and is used by 90 percent of all super computers. Because of its speeds the Draw It or Lose It software will run extremely efficiently.
5. **Distributed Systems and Networks**: A distributed system will allow the Draw It or Lose It software to be run on multiple clients (which are in essence all temporarily part of the same network) simultaneously which is of course a necessity considering this is a team-based game where multiple users will need to be playing the game at the same time. This system is critical to the functionality of the application.
6. **Security**: Linux is certainly not known for its high-level security. Considering the application at hand there is not a need for high level of personal information. I would strongly suggest that the The Gaming Room does not require a significant amount of information for an account to be created withing the Draw It or Lose It application because it is not necessary. With that in mind we will certainly ensure that the user create a username/password combination to create an account within the application. That password will have a certain restrictions and requirements to ensure that it is an effective password. We will use transit encryptions to secure those usernames and passwords. Additionally whoever we decide to go with for cloud storage will have to meet a certain number of security requirements in order to seal the job with The Gaming Room to ensure user security.