

Gaming Room

# **CS 230 Project 1 – Neal Goulas**

Version 1.1

## 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

[**System Architecture View**](#_ilbxbyevv6b6)3

[**Domain Model**](#_8h2ehzxfam4o)4

**Evaluation** 5

**Recommendations** 6

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 9/18/2022 | Neal Goulas | Initial Design Document Creation |
| 1.1 | 10/2/2022 | Neal Goulas | Client/ Server Update |
| 1.2 | 10/16/2022 | Neal Goulas | Added 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)

The Gaming Room is looking to develop a web-based game that serves multiple platforms based on their current game, Draw It or Lose It, which is currently available in an Android app only. A game consists of four rounds of play lasting one minute each. Drawings are rendered at a steady rate and are fully complete at the 30-second mark. If the team does not guess the puzzle before time expires, the remaining teams have an opportunity to offer one guess each to solve the puzzle with a 15-second time limit. If any other core features or mechanics are needed, The Gaming Room should follow up with what is needed after review of this document.

## [Design Constraints](#_2et92p0)

* A game will have the ability to have one or more teams involved.
* Each team will have multiple players assigned to it.
* Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.
* Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.
* Server/ Backend Capacity should be able to handle the traffic generated.
* Web versions should not be too different from Web.
* The proposed solution should be able to run cross platform via web-based implementation.

## [Domain Model](#_8h2ehzxfam4o)

The GameService class is a singleton where only one instance can exist. This makes it easier to keep trck of the game, team, and player ID’s. The GameService class can create a Game. A Game class can create a Team, and a Team class can create a Player. Game, Team, and Player are all inherited from The Entity Class. There can be many entities. The ID and Name are stored in the Entity class. The ProgramDriver class is where the main method is and the SingletonTester is a developer class to test the singleton.

**"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**

The table below shows the characteristics, advantages, and weaknesses of each operating platform (Linux, Mac, and Windows) as well as mobile devices.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | A Mac server can communicate with Mac clients very easily. It would be harder to communicate with other OS’s. Hardware for Mac would be more limited than other platforms. | Linux is a widely used server platform. It can easily communicate to other platforms. It is very secure. Linux is open source and cheaper than other options. | A windows server has a lot of the advantages of the Linux server. It is not open however so will cost more. | Mobile devices should not serve as servers since they have limited power and connectivity. |
| **Client Side** | Mac is more costly than other platforms. It is also a closed system where any applications must be approved by Apple. One big disadvantage is that you are only allowed to submit applications build by an apple device to Apple for approval. | Linux is open source and most cost effective however not many users are familiar with Linux. Often developers do not release Linux versions of applications due to the low number of users. Linux clients are more well suited to companies and enterprise users. | Windows is the most popular platform for both users as well as developers. It is less costly than a Mac system most of the time. Many users are familiar with Windows already. | Mobile devices are probably the most common client type today. The only disadvantages could be less powerful than a desktop platform and that applications have to be able to scale to smaller screen sizes. |
| **Development Tools** | Almost any development tools are available for the Mac platform. Some Mac systems are limited by graphical power. | Linux can be used to develop with many tools and languages. Some tools that are available on other platforms may not be available on Linux. | Windows is the most popular for development tools. Power for graphics is normally not an issue. You can use many tools and languages to develop on the windows platform. | Mobile devices are not ideal for development tools. Due to the lack of power and screen size, desktop platforms should be used for development. |

**Recommendations**

**Operating Platform:** For the server, Linux is recommended due to the lower cost and code agnostic requirements. The Linux server will be able to easily communicate with clients on multiple other platforms.

**Operating Systems Architectures:** With the current server setup being Linux, we are able to create a client on any platform and are able to connect to the server. If multiple client code bases are not wanted, a webapp that can be run on any device can be looked to as an option.

**Storage Management:** For storage, the server or an external database should be used. This will allow client builds to be smaller and any relevant info needed can be downloaded from the server. This also will allow to help prevent cheating since anything that comes from the server is trusted. This also allows for new additions without the need to push out new client builds.

**Memory Management:** Memory management will work close to the same on all platforms. If we were to implement the above storage method, it would go as follows for the image handling. An image would be pulled into memory from the server at this point. if desired, it could be saved locally to the drive of the client device. This would allow faster loading if the image was to be used again. Likely this would be a user option that could be selected in settings to let the user decide to use that drive space or not. Once in memory, the image can start to be displayed. After the round is complete, we can clear the image from memory since it is not needed. This approach allows to keep the memory allocation for the image usage low and allow for more overhead with other features. The one thing we would not want to do is to keep the current needed image active in memory. As most devices today could handle the 1.6 gigabytes it would take to keep all the images in memory, some lower end mobile devices could have issues as some only have one to two gigabytes total memory to work with.

**Distributed Systems and Networks:** Since the image database will be shared with all the clients, the clients will interact with the server only. This will be a client-server model. Clients will contact the server where, once authorized, can request information from the server and use it as needed.

**Security:** Since the clients will only communicate with the server, no other clients will have information about other clients. The server will also authenticate each client with a user name and password, only giving access to approved information based on the users role and authorizations.