

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 2.0

## Table of Contents

[**CS 230 Project Software Design Template**](#_heading=h.gjdgxs)1

[**Table of Contents**](#_heading=h.30j0zll)2

[**Document Revision History**](#_heading=h.3znysh7)2

[**Executive Summary**](#_heading=h.2et92p0)3

[**Design Constraints**](#_heading=h.tyjcwt)3

[**System Architecture View**](#_heading=h.3dy6vkm)3

[**Domain Model**](#_heading=h.1t3h5sf)3

[**Evaluation**](#_heading=h.2s8eyo1)3

[**Recommendations**](#_heading=h.3rdcrjn)5

## [Document Revision History](#_heading=h.3znysh7)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 03/20/22 | Alyssa Mazzucotelli | Created Document |
| 2.0 | 04/10/22 | Alyssa Mazzucotelli | Evaluation added |
| 3.0 | 4/22/22 | Alyssa Mazzucotelli | Recommendations added |

## [Executive Summary](#_heading=h.2et92p0)

The Gaming Room would like to develop a web-based game for multiple platforms based on their current game, Draw It or Lose It, which is currently available only for Android.

## [Design Constraints](#_heading=h.tyjcwt)

* Multiplatform: This will probably mean multiple builds of the game will be necessary depending on what platform is able to run what.
* One or more teams: Multiple teams will need to be implemented with multiple players on each team.
* Unique game and team names: The users must be able to check whether a name is already in use when choosing a team name.
* Unique identifiers: Only one instance of the game is allowed in memory at a time.

## [System Architecture View](#_heading=h.3dy6vkm)

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](#_heading=h.1t3h5sf)

In the UML class diagram below we have the public class ProgramDriver, which contains the main() method that will run the game. ProgramDriver also uses the class SingletonTester which contains the method testSingleton().

The Entity class is a base class for others that contains private elements of a long named ‘id’ and a string named ‘name.’ These can be accessed but not changed by other classes through the public methods getId() and getName(). Entity also contains a default method for displaying an element called toString(). Entity also contains a private constructor as well as a public one that takes the parameters of a long and a string named id and name respectively.

Inheriting from Entity are Game, Team, and Player. All three of these include an id and a name as they extend from Entity. Player contains no new variables but does contain the public class of Player which takes the parameters of id and name to create. Player also contains another toString() method to display an element.

Team similarly inherits from Entity but has the added variable of a private list consisting of Player classes named players and has a 0 to many relationship with the Player class. Team also has the ability to add a player to the list through the use of the addPlayer() method which takes a string name of a player to add the player to the list of players. Team can also create a Team with an id and a name.

Game in turn has a 0 to many relationship with Team and has a private list of Teams. Game does the same thing as Team did with Player but now with Team instead and makes a list of Teams named teams.

GameServices then has a 0 to many relationship with Game but unlike the previous three does not directly inherit from entity though it does through Game. GameService is able to hold many instances of Game which are able to hold multiple Teams which can hold multiple Players without needing multiple GameService instances. GameService also introduces a few more longs including nextGameId, nextPlayerId, and nextTeamId which help it iterate through all three groups. There is also a private instance of GameService known as service which is part of the singleton pattern with getInstance() to make sure only one instance of the class exists in the memory at any time. GameService also contains different public methods to access the game through Id or name, the game count, the next player Id, and the next team Id as well as to add a game.

**"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](#_heading=h.2s8eyo1)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | **Characteristics:**  Mac hosted website servers will run across any OS  **Advantages:**  -High security  -Nice GUI  **Weaknesses:**  -Not great for mixed OS environments  -Limited usability and functionality  **Price:**  Starting at $19.99 for server management app | **Characteristics:**  Linux hosted website servers will run across any OS  **Advantages:**  -High security  -High configurability  **Weaknesses:**  -Complex for those not familiar with Linux already  -Updating more complex  -No long term support  **Price:**  Free | **Characteristics:**  Windows hosted website servers will run across any OS  **Advantages:**  -Beginner- friendly  -Easier to update  -long term support  **Weaknesses:**  -Low security  -High vulnerability  -Resource intensive  **Price:**  High and increases with users | **Characteristics:**  Most mobile devices can be used to run servers as well.  **Advantages:**  -It’s possible  **Weaknesses:**  -Mobile device is unable to function as a mobile device while hosting.  **Price:**  The price of the phone that needs to basically be sacrificed for this |
| **Client Side** | **Cost:**  Up to $999 for unlimited client license  **Time:**  A short amount of time needed to convert the client from android to Mac  **Expertise Level:**  Mid-level | **Cost:**  $349 per year license  **Time:**  A short amount of time needed to convert the client from android to Linux  **Expertise Level:**  Mid-level | **Cost:**  $501 for small businesses  **Time:**  A short amount of time needed to convert the client if it does not already run on Windows  **Expertise Level:**  Mid-Level | **Cost:**  $99 a year for the Apple Developer Program + a percent of all sales  **Time:**  Moderate amount of time needed to convert the client from android to iOS  **Expertise Level:**  Mid-Level for web-based or high level for an iOS app |
| **Development Tools** | **Programming languages:**  HTML/CSS, JavaScript, Python, Java, SQL, C#, PHP, C++, C, Ruby  **IDEs:**  Visual Studio, Xcode, CLion($89 per year), AppCode($89-54 per year), PhpStorm($89 per year or $19.99/month), Atom | **Programming languages:**  HTML/CSS, JavaScript, Python, Java, SQL, C#, PHP, C++, C, Ruby  **IDEs:**  Visual Studio, Xcode, CLion($89 per year), AppCode($89-54 per year), PhpStorm($89 per year or $19.99/month), Atom | **Programming languages:**  HTML/CSS, JavaScript, Python, Java, SQL, C#, PHP, C++, C, Ruby  **IDEs:**  Visual Studio, Xcode, CLion($89 per year), AppCode($89-54 per year), PhpStorm($89 per year or $19.99/month), Atom | One of the previously mentioned OS and their tools would be needed to develop for a mobile device as well as deploying it to the app store of choice |

## 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**:

I would recommend that The Gaming Room use Red Hat Enterprise Linux to expand Draw It or Lose It to other computing environments. Linux would have a mix of the lowest cost and high security and configurability. I recommend the Red Hat Enterprise Linux as it is fairly cheap while also offering the option to upgrade with add-ons including management, different storage systems, and extended update support.

1. **Operating Systems Architectures**:

The architecture of Linux consists of several layers. The innermost level is the hardware that consists of the memory, CPU, arithmetic logic unit, storage, I/O, and other physical devices. The next level up is the kernel which manages the system’s resources. The next level is the shell that provides the user interface and encases the insides of the kernel from the user. All of that combined creates the application.

1. **Storage Management**:

Linux servers have a System Storage Manager as well as Red Hat having the available update for Resilient storage. Linux server storage systems can also become more complicated by using Device Mappers, Logical Volume Managers, and Multiple Devices.

1. **Memory Management**:

Linux uses a hierarchical file system structure. Keeping files organized is already a good practice in general but it will be helpful for Draw it or Lose it as they have a large amount of pictures that the game will choose to display. It is important that once the game is done with one picture that it removes it from its memory so as not to slow down over time.

1. **Distributed Systems and Networks**:

The Gaming Room would like to have Draw It or Lose It work across various platforms so what better match than Linux, the OS that can run anywhere and is compatible with all major cloud services. The only main thing that The Gaming Room would need would be an iOS specific app client assuming they would like to put it in the App Store. All clients across all platforms would have authenticated access to the same server through the log in system. The only time outages would be a problem would be when the chosen cloud service had its own outage which is not really anything that The Gaming Room could predict or control.

1. **Security**:

The Gaming Room is already on track to having a security system built into their log in system. Each user is assigned a different tag or role and then is only able to access certain designated information based on that role. I would recommend some form of encryption for those roles though as it may be easy for a hacker to trick the server into thinking they are an admin if all they need to do is change their role from user to admin to gain access to everything.