

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.2

## 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)3

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/11/22 | Zac McBride | Set up environment for the Draw It or Lose It game |
| 1.1 | 11/25/22 | Zac McBride | Evaluate choices for OS to host server |
| 1.2 | 12/5/22 | Zac McBride | Recommendations for system |

## [Executive Summary](#_sbfa50wo7nsh)

The web-based version of the game: Draw It or Lose It will need an environment set up for players to play the game. Using unique names for the game, team, and player, this environment will allow only one instance for each of these.

## [Design Constraints](#_2et92p0)

Because this is a web-based application, the functionality will be designed in the Java programming language. It will save a list of games that are being played. Each game can have one or more teams consisting of multiple players per team. Every player name needs to be unique so that only one instance of a player name can exist. Every team name will likewise only have one instance of a specific name. The name of each game will also be unique so that only one instance of each game will exist in memory. Upon creation of player, team and game names the environment will need to check that that name is not currently being used under that category.

## [Domain Model](#_8h2ehzxfam4o)

GameService is a singleton class that creates a list of game IDs. This instance will be created under the service attribute. The other attributes will allow for the next game, next player, and next team to be stored as a long value. The main function is found within Program Driver and will be able to access the encapsulated attributes of the GameService class by adding a game to the list, getting a game from the list, and getting the next player and team. The SingletonTester class is used to test that there is only one instance of the GameService by trying to create and access another GameService instance.

There is a parent class called Entity. The Game, Team, and Player classes all inherit from Entity as they have common attributes including a long value for the id and a string that stores the name of each entity. They all share methods that allow these attributes to be accessed by outside classes. All three of these classes are polymorphic in that they override certain functions of the Entity class particularly with the constructor method for each class. The Team class stores a list of Player class objects, and the Game class stores a list of Team class objects. This allows for players to be a part of a team and teams to be a part of 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](#_2o15spng8stw)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Stable, higher cost for licensing, GUI to simplify administrative process | Can handle large amounts of traffic and complex activity,  low-cost open source, stable, difficult to learn and use | Easier interface, more expensive proprietary costs for licensing, less stable, GUI to simplify administrative process | Wide variety of devices makes for variety of characteristics, advantages, and weaknesses, can use open-source software for server |
| **Client Side** | Higher cost, easier to use if familiar with Mac OS, | Low cost, more expertise | Higher cost, easier to use, | Cost dependent upon device, |
| **Development Tools** | Programming Languages: Swift, HTML, Java, JavaScript  IDE: XCode, Eclipse, Visual Studio | Programming Languages: HTML, Java, JavaScript, C++  IDE: Eclipse, Visual Studio | Programming Languages: HTML, Java, JavaScript, C++  IDE: Eclipse, Visual Studio | Programming Languages: HTML, Java, JavaScript, Swift (On iOS Devices)  IDE: Android Studio (Android Devices), XCode (iOS Devices) |

## Recommendations

1. **Operating Platform**: Though a Windows operating platform may be easier to use, a Linux storage system will be the best option as it will be the most cost effective and provide the security required to run the server.
2. **Operating Systems Architectures**: A server based on the Linux Kernel such as Ubuntu or CentOS. One of these will provide a very secure server and they tend to be more stable than a Windows server would be. Linux servers use a command line interface rather than a GUI which may be more difficult for some to use but may give more flexibility.
3. **Storage Management**: The storage system will need to be capable of storing at least 2 GB of data which will include the code for the system as well as the high-definition photos that will need to be rendered during gameplay. It would be best to use index allocation while storing files as this will be fast and reduce fragmentation. A cloud-based storage bought from another company would also be a very cost effective storage option.
4. **Memory Management**: The Linux Kernel has a complex memory management system. Some of these features include a virtual memory primer which saves physical memory by mapping physical memory pages to virtual addresses. Compaction reduces fragmentation by moving pages to allocate contiguous area of physical memory.
5. **Distributed Systems and Networks**: Using a client-server architecture will make it easier to allow the game to be used on multiple platforms. This will allow multiple kinds of devices to connect to the server via an internet browser. Linux Kernel server are easy to scale depending on the demand of the server. This will allow for an increase in users. Linux server are more stable which will allow for better connectivity.
6. **Security**: Using a Linux Kernel will increase security in itself but the client-server architecture also allows for authentication and authorization to be a part of the security of the system. Users will log in and only be allowed the necessary access to accomplish the tasks they will perform. Designing the system to have as small a footprint as possible will increase security as well.