

Draw It or Lose It

# **CS 230 Project Software Design**

Version 3.0

## Table of Contents

[**CS 230 Project Software Design** 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 5**

[**Recommendations 7**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 01/26/24 | Jasper Conneway | Start Software Design Document |
| 2.0 | 02/08/24 | Jasper Conneway | Evaluation |
| 3.0 | 02/23/24 | Jasper Conneway | Recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room, a client of Creative Technology Solutions (CTS), wants to develop a web-based game that serves multiple platforms based on their current game, Draw It or Lose It. They need a game application that can store unique games with unique teams. They want users to be able to check whether a name is in use when choosing a team name.

## Requirements

* A game can have multiple teams.
* Each team will have multiple players.
* 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.

## [Design Constraints](#_2et92p0)

Develop a game application in a web-based environment.

Use the Singleton Pattern to develop a game application that only creates one instance of a game.

Use the Iterator Pattern to check if a game, team, or player name already exists in memory.

## [System Architecture View](#_ilbxbyevv6b6)

There will be a base Entity class that contains the attributes ‘id’ and ‘name’ to signify that each game, team, and player has their unique id and name.

There will be a game class to create a unique game instance.

There will be a team class to create a unique team instance.

There will be a player class to create a unique player instance.

There will be a program driver class to test the game application and a singleton tester to ensure each instance is unique.

The game, team, and player classes will inherit from the base Entity class.

## [Domain Model](#_8h2ehzxfam4o)

The base Entity class is the Super Class of the classes Game, Team, and Player. The Entity Class holds each entity's identifier and name. Each class that inherits from this class can get the id and name of their respective entity. The Game class will use the Team class to create a list of teams for a game. The Game class can add a team to a game. The Team class uses the Player class to create a list of players for a team. The Team class can add players to a team. The Player class defines what a player is by giving them each an id and name. (It doesn’t appear that a player’s name must be unique?).

The GameService class is associated with the Game class as it creates a list of Games. Just as the Game class is associated with the Team class, the Team class is associated with the Player class.

The ProgramDriver class is the only class with a main() method which tests the Game Application. This class uses the SingletonTester class which shows the Game Applications behavior.

**"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** | MacOS is an operating system created by Apple. It has a streamlined user interface and a simple graphical user interface. The operating system and hardware are consistent and run efficiently. MacOS is less likely to have security issues or vulnerabilities. It’s also great with multitasking and fast switching between tasks. There are some trade-offs with using MacOS. One being the cost as they are expensive and require an Apple/Mac device. It is illegal to download MacOS on a device that is not made by Apple. Hardware upgrading is limited and difficult but necessary for the devices lifespan. The games are also limited and the OS is not made for certain games. | Linux is an operating system that is open-sourced and reliable. It is less likely than others to crash or freeze. It also has very good security because of the community of programmers using it. The hardware is supported on multiple operating platforms. The trade-off of Linux is how hard it may be to learn the operating system. Someone new to programming would need time to understand the system. Some games are not available for Linux and technical support is limited. One would likely have to rely on online resources for professional help. | Windows is an operating system made by Microsoft. Windows is used by a large population and has more technical support. It can be more convenient as the hardware is not restricted to a specific device. There are some disadvantages to using Windows. It has a lot of security attacks. There are exploits for hackers to utilize to break into a Windows system. It can also be costly for programming as it is not open-sourced. | Mobile Devices are portable devices that one can carry from place to place. These devices range from cell phones to smartwatches to tablets. They can be handheld, and some are wearable. These devices have high processors and RAM.  There are some disadvantages like needing internet or Wi-Fi and relying on battery life. Security is also an issue as it isn’t as secure from attacks. They’re also not cost-efficient. |
| **Client Side** | The game application would have to be available in Apple store and compatible with MacOS. Cost would include an Apple device such as a Mac. This is because it’s better to develop software for Mac on its own hardware. Cost and Time depend on how long it takes to develop the application on a Mac system.  Lastly, licensing costs would need determined and added. | The game application would have to be compatible with Linux. Consider how long it would take to test the code as it needs to be standardized and accepted coding style. Security concern for Linux is the least of the four OS. Cost and Time depend on how long it takes to develop the application on a Linux system. Lastly, licensing costs would need determined and added. | The game application would have to be compatible with Windows. Security would be the big issue here for cost and time. The application would need to be checked regularly for malware and security attacks. Lastly, licensing costs would need determined and added. | The game application would have to be compatible with Mobile Devices and restricted to specific devices. Smartwatches would not need access to the application. Battery life is an issue here. Use should be limited to Wi-Fi or should not impair the battery harshly.  It would need to adapt to a smaller screen. Lastly, licensing costs would need determined and added. |
| **Development Tools** | There is not any specific IDE or language required for developing software on Mac. Some of the best IDEs to use include Visual Studio Code, PyCharm, Xcode, and more.  Tools are the same as Windows, Linux, etc., but one is Docker Hub and another is Atom. Languages include Python, MATLAB, and Swift. | The best IDEs for Linux are Sublime Text, Atom, Eclipse, and more. The best languages for Linux are C/C++, Java, and Python. As for tools on Linux, Docker Hub and Atom are also as good as it is for Mac. | The best IDEs for Windows are Visual Studio Code, IntelliJ IDEA, and PyCharm. The best languages include C#/C++, Python, and Java. As for tools on Windows, some are Atom, Delphi, Axure, and Jira. | The best IDEs for Mobile Devices are Android Studio, Xcode, Visual Studio Code, and more. The best languages are Java, Kotlin, Swift, and C#. As for tools, some are Android Studio, Xcode, and AppCode. |

## Recommendations

1. **Operating Platform**:

Based on the evaluation of what is necessary for the Draw It or Lose It application to function properly and successfully, it is recommended that the Windows Operating System be used. The Operating Platform will not restrict the game to one computer environment and users won’t have difficulty utilizing the platform to play the game. It does not require much expertise of its platform and the cost can be manageable.

1. **Operating Systems Architectures**:

The Windows Operating Systems Architecture recommended is the hybrid kernel model. This architecture is used by most of the new versions of Windows. The hybrid kernel model is a compromise between microkernel and monolithic kernels. This means Windows uses a microkernel for the core services and a monolithic kernel for the drivers and subsystems. This provides Windows with flexibility and reliability. Using the hybrid architecture will ensure our Draw It or Lose It application is highly performing or outperforming other Operating Platforms. The hybrid architecture will also provide security for our application.

1. **Storage Management**:

It is recommended that for the Draw It or Lose It application we use Cloud storage. Cloud storage backs up its data off-site, on remote servers. It allows a user to gain access to their storage from any computer. This will allow our users to transfer their game data from one device to another. Cloud storage is also typically less expensive than hardware storage and can be upgraded for a higher capacity.

1. **Memory Management**:

It is recommended that the Draw It or Lose It application utilizes the Garbage collection to allocate memory. The garbage collection will dispose of objects not in use and create space for new objects. It is also recommended that we use the caching technique. Caching can store image files that are called upon frequently and can speed up the program when looking for a common image.

1. **Distributed Systems and Networks**:

Distributed Systems and Networks are useful in connecting different devices to the same game. The Game Room would like its application to be accessible to various platforms and for users on different platforms to have the ability to communicate. It is recommended to use a RESTful API to accomplish this. REST is ideal for web-based gaming applications and allows interactions between different platforms. It does this by having the server-side listen for a request from a client. Then the server will send a resource to the client for them to manipulate. The resource sent to the client allows the client to manipulate some part of the application.

1. **Security**:

There are several ways we recommend protecting the user and their information. First, we recommend using authorization and authentication. Each user should have set privileges and require logging in to gain access to their account. Users can also use multi-authentication, but this should be provided to the user as an option. Second, the application needs to be routinely checked for cyber threats like malware and the application should be kept up to date to help prevent attacks. Outside vendors can be utilized by pen-testing the application for any vulnerabilities.