

<Draw it or Lose it>

# **CS 230 Project Software Design Template**

Version 1.1

## Table of Contents

[**CS 230 Project Software Design Template** 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 4**](#_Toc115077325)

[**Recommendations 5**](#_Toc115077326)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0  1.1  1.2 | <05/25/2024>  <06/9/2024>  <06/23/2024> | <Sarah Wu>  <Sarah Wu>  <Sarah Wu> | <Finished product after Entity class construction>  <Finished development requirements table>  <Finished Recommendations> |

## [Executive Summary](#_sbfa50wo7nsh)

With this document we are hoping to highlight and detail our solution for the Draw It or Lose It game. Draw It or Lose It is based off the 1980s television game show Win, Lose or Draw. In this game, teams would compete to guess which image is being drawn. In the software version of this game images will be rendered and teams will compete on a time limit to guess which image is being shown. We will develop this web-based game to be compatible with multiple platforms allowing users on many different devices to enjoy it.

Our goal is to provide fully available software by allowing seamless cross-compatibility across many platforms. We also will develop the game while keeping the ultimate user experience in mind. Most of all we are looking to create a game filled with a fun and enjoyable time for all.

To do this, we will need to focus on client requirements. We will also need to ensure we determine our design constraints and focus on how to remedy those. Most importantly we will need to ensure the accuracy and reliability of our product.

## Requirements

1. Web Based Game-The game should be cross compatible with multiple platforms.
2. A game will have the ability to have more than one team involved.
3. Each team will have multiple players assigned to it.
4. Game and team names must be unique. Users should be able to check whether a game or team name is already in existence before creating a new one.
5. Only one instance of the game can exist at a time.

## [Design Constraints](#_2et92p0)

Web-based platform- This game must be developed on a web-based platform. This will ensure cross-compatibility across multiple devices and browsers. Programming language and compilation must be chosen with this in mind.

Multiple teams- The game will have the option to have multiple teams playing at once. Multiple teams allow players to compete against each other in games.

Unique Game, Team, and player names- For the ultimate experience all instances of games, teams, and players must be unique. There needs to be a way for players to check if an existing game, team, or name exists and if it doesn’t to create a new instance of that.

Unique Game Instances-Only one unique game can be running at a time. This makes sure that there is no duplication happening which could cause conflict between the same game names. All running games must be unique.

## [System Architecture View](#_ilbxbyevv6b6)

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](#_8h2ehzxfam4o)

This UML diagram displays the architecture for our web-based application. Each created class is depicted below with all associated attributes and functions. To start we have created our Base class Entity. It creates our private attribute id and name. The subclasses of Entity are Game, Team, and Player. Each of these classes extend off of the Entity class and Inherit attributes. This means they will inherit the id and name attribute to be used inside of their class. These subclasses include functions to add new players and teams to the game.

The GameService class has an association with the Game Class, which has an association with the team class, which has an association with the player class. We can see from all of these that each of these classes can have 0 or more instances with each other. The ProgramDriver class holds our main() function which has a directed association with our SingletonTester class.

We can see many object oriented programming principles at play here. As noted early we can see inheritance in the way the Game, Team, and Player classes directly inherit attributes from our super class Entity. We can see Polymorphism in the allowance of method overloading, which is where classes can perform actions with the same method name but using different attributes or code. One example of this is in the GameService class we have two methods with the getGame() name, but they are looking for games with different attributes(name: String, and id: long). Abstraction and Encapsulation are also important object oriented principles because they seeks to protect data and make it readily available as well when needed. Many of our attributes are set to private notated by a “-“ in the diagram. Getter and setter methods are created to access or modify those attributes to control the modification of this data. Also with Abstraction separate classes are created to contain specific methods and instance hiding the complicated processes away from the main method.

**"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)

Using your experience to evaluate the characteristics, advantages, and weaknesses of each operating platform (Linux, Mac, and Windows) as well as mobile devices, consider the requirements outlined below and articulate your findings for each. As you complete the table, keep in mind your client’s requirements and look at the situation holistically, as it all has to work together.

In each cell, remove the bracketed prompt and write your own paragraph response covering the indicated information.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac servers are easy to configure and use. They offer fast deployment processes and are less vulnerable to malware and viruses. On the negative side, Mac servers have higher hardware requirements and limited scalability for large environments. | Some advantages to Linux servers are they are secure, reliable, backwards compatible, and cheap. The biggest disadvantage to Linux servers is the steep learning curve and the amount of time required to configure and maintain. | Windows servers are very secure, scalable, and have comprehensive management tools. They generally come with GUI which makes them easier to use. Negatives include high cost and licensing and potential vulnerability to malware. | Mobile devices apps allow automated updates, scalability and enhanced security. They are easier to maintain and in general mobile apps are more efficient and run faster. Maintenance is more cost effective. On the negative side both iOS and Android devices will need separate considerations. |
| **Client Side** | Mac Clients offer a  UI that is in general user friendly and intuitive. On the negative aspect, Mac hardware is expensive. Many users may not use Mac devices and may need training on how to best access and use. Macs use proprietary software and sometimes hide technical features. | The benefits of using Linux clients is the ability for high customization, free license, and low hardware requirements. The biggest disadvantage to Linux is that it requires high user knowledge to operate and setup. Also Linux is not standardized. | Windows is compatible with a large range of different applications and software. They provide user-friendly UIs. Negatively, Windows licensing cost is expensive as it requires licenses per device. Windows also has high resource requirements. | There are many different mobile device manufacturers and mobile devices sizes and types including phones and tablets. Client software will need to be designed for various UI sizes as well as different types of access features like gestures, hard and soft keyboards, and different security limitations. |
| **Development Tools** | Macs can run most programs that are compiled for Mac. Popular programming languages include java and python. Popular IDEs include VS Code, Visual Studio, and Xcode. | The most common Linux programming languages are C and C++, and Python. There are many IDEs for Linux including CLion, Eclipse, and Visual Studio. | Many programming languages are used with Windows including C#, Java, and Python. Popular IDEs include Visual Studio, Eclipse, and Pycharm. | Mobile apps use Java, Kotlin, and C#. There are many IDEs for mobile app development including Android Studio, Xcode, Visual Studio, Eclipse, and NetBeans. |

## 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**: Given the requirements for the Gaming Room to operate across multiple platforms and connect multiple users I believe the best operating platform to use would be a Linux based cloud platform, like AWS. A cloud-based platform is easily accessible by many users. Another important aspect of cloud-based platforms is the ease of scalability and flexibility. We can easily expand based on our current and future needs.

Using a Linux cloud-based platform will also allow us to better disaster recovery. Data is stored remotely and can be made redundant. Data backups can be made regularly to provide data recovery options. Also, our overall costs will be easier to anticipate as there will be a steady fee. If we attempt to host our own servers we will have unanticipated hardware, infrastructure, and personnel costs.

1. **Operating Systems Architectures**: By choosing a cloud based operating platform like AWS, we can use a hybrid architecture. A hybrid architecture will allow us to use multiple architectures based on need. This layered approach will be easier to manage and allow us to customize our protection and increase our security. We can use the wide area network to combine multiple devices and networks to our main cloud service. This architecture will make our game the most scalable and flexible.
2. **Storage Management**: If we choose a Linux-based cloud platform we will also be using a remote hosted cloud managed storage solution. We can use this service to build redundancies by making sure there is also an automated backup of all our user-data and important data as well. We can build a disaster recovery system to minimize our downtime. Using can use a hybrid cloud storage service to have access to infinite amounts of cloud storage.

Hosting our storage remotely will allow our users to access data from different regions and platforms. Data can be accessed, modified, and retrieved and lightning-fast speeds using the cloud’s high-speed cache. We also won’t have to worry about expensive hardware costs or physical security for our data, as all these costs will be taken care of as part of our remote hosted package.

To keep costs low we should choose an efficient, but high-quality image file type like JPEG. This will ensure we are only using a necessary amount of data. We will also need to balance our redundancy needs with our scalable costs to determine the optimum amount of data we need to purchase.

1. **Memory Management**: Using a cloud-based solution we can specify the memory allocation needed for various functions and applications. Generally, memory usage is scalable and costs for use increase the more resources are used. We will want to optimize our memory usage to provide a seamless user experience, but also not use too many resources to keep costs low.

As our game needs to constantly process over 200 high-definition images, I recommend we compress these images to the smallest file size possible while keeping the visual quality high to the end-user. Memory should be cached to store some of the image data, without images needing to be reloaded on each access. Images should also only be loaded on an as needed basis.

1. **Distributed Systems and Networks**: Using a Linux based cloud platform means our users will be able to connect to the same central hub to access data. We will use a transit gateway for routing connections between our main hub and on-premises networks. We can use a cloud server to host multiplayer games with matchmaking options. If we use elastic computing solutions, we can have the resources increase during peak player times and decrease in low play times to keep latency low.

Depending on location, many cloud based solutions offer regional availability for gaming servers as well. Games can originate from anywhere globally, but depending on region matchmaking sessions will attempt to find local matches first to connect to local cloud-based game servers, reducing latency. Traffic will be routed between clients and cloud server directly connecting all regions to one central hub.

Sessions can also be managed between like devices for fairness between different UIs. End-users on mobile platforms can connect with other users on the same platforms. Likewise, those using personal computers can connect to others using personal computers. This keeps games on a like for like basis.

1. **Security**: Using a cloud-based platform will mean that physical security is already taken care of. We will not need to provide our own physical security as this will all be managed by the remote hosted service. Cloud-based platforms are architected to be extremely secure as they connect users from all over the globe to one central location. Logging and monitoring services can be enabled to make sure any threats are tracked in real time. User permissions and identities will be set to make sure the data is secure and protected. Authentication should be enabled with MFA to ensure that only those who are supposed to be accessing the data are. Game data and services should be automatically patched and updated to stay current to prevent chances of infection and breech.