

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## 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 | 5/24/2024 | Tyler Doupe | Prepare software design document and begin developing the Draw It or Lose It |

**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 wants to develop a web-based game that serves multiple platforms with their current game, Draw It or Lose It. The Gaming Room would like their game to support one or more teams with multiple players assigned to each team. They also would like to integrate unique team names and game names. In order to achieve this, the use of the Singleton pattern will be integral to implementing single instances of games. The game will also need to ensure that the team names and player names are unique.

## Requirements

The first requirement that The Gaming Room demands is for their game to serve multiple platforms. They also requested that their game supports multiple teams with multiple players on each team. In addition, The Gaming Room requested for team names, game names and player names to be unique by only allowing one instance of each.

## [Design Constraints](#_2et92p0)

Developing Draw It or Lose it in a web-based distributed environment presents several design constraints will need to be address. To begin, with multiple teams and players active on the game at one time, especially in a pace-based game such as this game, we need to consider the potential for network latency. Although we can not control all components of the latency players may exhibit based off how they are connecting to our web-based game, it will serve as a constraint that will affect the user’s play and ultimately their pleasure with the game. We also will need to handle the scale at which the game is played. Depending on the number of teams, players and overall active users, we may receive a large number of requests. The web-based game needs to be capable of handling a potentially large number of requests all at the same time. Having unique names and games will demand data security as users will be required to input personal information to create their own unique profile and team that they do not want shared to the public. Lastly, another design constraint that should be considered is browser compatibility. The game will need to be playable from numerous browser types if we are to make the game truly multi-platform and appeal to a larger number of users.

## [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)

Entity – The Entity class is the base class. It is the building block in which all of the other classes fall under. It identifies the id and name.

Player – The Player class breaks out each player in the game. Each player appears to have a name and an id associated. They also have an associate Team which has its own class and associated attributes.

Team – The Team class holds a list of players associated with the Team. It has attributes of id and name, along with an addPlayer attribute.

Game – The Game class contains a list of teams to play in one game instance. It holds the attributes for the game name and id as well as the ability to add teams to the game.

GameService – The GameService class appears to hold all the essential information to run the game. The GameService class manages the games list, assigns the next game id, the next player id, and the next team id which implements the unique game, player and team names that The Gaming Room desired. Through the use of the Singlton pattern, it enables the single instance of a game. There are also a list of attributes associated with the GameSerice class such as adding a game, getting a game, getting a player, or getting a team.

The Game class ultimately inherits all of the attributes of the GameService class to allow for the game to function and run with a one-to-many relationship. In addition, there is a one-to-many relationship between the Game and Team classes which allows for Teams to be created with their own unique name and id. This structure is seen again as there is a one-to-many relationship between the Team and Player classes which allows multiple players to be on one team. The entity, base class, inherits all of this which ultimately allows for the game to function as The Gaming Room requested.

**"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** | Characteristics:  Mac offers flexible terminal commands  Advantages:  Strong built-in security and development tools  Disadvantages:  Costs more for licensing and is less commonly used for hosting web applications | Characteristics:  Open source and widely used for web servers.  Advantages:  More cost friendly. Enhanced security.  Disadvantages:  Requires more technical expertise. | Characteristics:  Very user friendly and popular amongst population.  Advantages:  Has a command prompt. Commonality provides comfortability. A lot of additional resources available.  Disadvantages:  More susceptible to viruses. Additional licensing costs. | Characteristics:  Most commonly used as a client rather than a hosting server  Advantages:  Mobile/portable.  Disadvantages:  Dependent upon mobile device OS updates and security. Limited processing power available. |
| **Client Side** | Requires expertise with Mac OS. It would be costly due to Mac hardware requirements. The time to develop would require a moderate amount of time. | Requires strong knowledge and expertise with Linux. The cost would be minimal. Would require a large amount of time and expertise to complete. | Requires moderate experience with Windows OS. Costly depending on licensing and tools required. Moderate time needed to develop. | Requires expertise and proficiency with mobile devices. Low to moderate cost which is dependent on what tools are used for development. Moderate to large amount of time required to develop. |
| **Development Tools** | Programming Languages:  Swift, HTML, CSS, JavaScript  Tools:  PyCharm, Eclipse, Notepas++, Visual Studio | Programming Languages:  Python, PHP, JavaScript, HTML, CSS  Tools:  Docker, GCC, Visual Studios, Eclipse | Programming Languages:  Python, JavaScript, HTML, CSS, Java, C++  Tools:  Visual Studios, Eclipse, Postman, CodeLite, PyCharm, MongoDB, React | Programming Languages:  Java, Swift, JavaScript HTML, CSS  Tools:  ReactNative, Xcode, Visual Studios, Eclipse |

## 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**:

With the objective being to expand Draw It or Lose It to other computing environments, I would recommend that we design the game to be a web-based game. Mac, Windows, Linux, gaming systems, IOS and Android devices and many others are capable of accessing the internet and process a web-based game designed with HTML, CSS and JavaScript. Although there will be additional considerations to be placed in the design process that would optimize the appearance and operability of the game on many devices, the devices will be able to run the game application thus the objective of expanding the game to other computing environments would be accomplished.

1. **Operating Systems Architectures**:

When considering the operating platform architectures, we should consider a multi-tier architecture as it would separate the data, application and presentation tiers to allow for a better understanding for how the application is operating by the defined layer. The presentation layer would be understood as the user end view of the game in action. The application layer would be the behind-the-scenes action to allow for the game to operate and provide the user the desired screen to use the game. The data tier would be the data storage and management that is necessary for the game to successfully operate. In addition to the multi-tier architecture, we would see the client-server architecture as this game would require real time interaction.

1. **Storage Management**:

With the decision to proceed with the game as a web-based game, we need to consider how and where the game’s data will be stored and managed. Rather than spending an excessive amount of money on servers to store this information, it would be best to invest into a cloud-based environment as it would allow for the game data to be pulled seamlessly and would save The Gaming Room a lot of money. Not only would the data be easily accessible for the game’s functionality by going through a cloud-based service, there would also be security measures already implemented into holding this data and would enable the game to be more scalable as th4e team’s developers can focus on updates for the game over time rather than worrying about the backend aspect and devising ways to manage the data required for the game to operate.

1. **Memory Management**:

When considering the memory of the game, it would behoove us to use smaller image files, efficient coding methods and instilling caching methods into the game. This would allow for quicker loads in the game, less bugs and errors to occur and ultimately smoother game play to occur. By making use of the memory techniques identified, the game will operate smoother and faster which will enhance the overall gameplay for the user and reflect positively on the game and company.

1. **Distributed Systems and Networks**:

With the game communicating between various platforms, we could instill a load balancing system that would ensure that each user connects to the server that is located closest to them. This would prevent lagging/connectivity issues and outages. It would also allow for the servers to be better distributed for the number of users per server. We also could make use of API services which would provide single entry point for all client requests where they would then be routed to the appropriate service to handle the request. Additionally, we could implement connectivity reattempts and timeouts that would allow for the client to continue seeking to connect to the server to attempt to reconnect quickly.

1. **Security**:

Security is a must have for the client and poor security can also tear The Gaming Room’s reputation down. In order to protect user information on and between various platforms we should first implement encryption and multi-factor authentication into our user logins and game communications. Multi-factor authentication has become increasingly popular, especially over the past few years, and could greatly enhance our security to ensure the user is who they say they are. Adding encryption into communications will only heighten our security. Additionally, the developer team should be constantly working on patches and updates for the game to stay up to date on security weaknesses.