

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0

## 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 | 07/14/22 | William Forbes | Added summary, constraints, and domain model. |
| 2.0 | 07/30/22 | William Forbes | Updated Evaluation |

**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 looking to develop a web-based, multi-platform game based on their current game titled “Draw It or Lose It,” which is only currently available on Android. Creative Technology Solutions will facilitate the development of the web-based version of the application and address the software requirements.

## [Design Constraints](#_2et92p0)

* Each game can have one or more teams.
* Each Team contains multiple players.
* Game consists of 4, 1-minute rounds.
* Drawings are rendered at a steady rate and completed at the 30-minute mark.
* If Team does not correctly guess before timer expires, remaining team has 1 guess within 15 seconds.
* Game and team names must be unique.
* Users can check if team name is in during name creation.
* Only one instance of the game can exist in memory at a time.
* Unique identifiers must be given for each instance of a game, team, or player.

## [System Architecture View](#_ilbxbyevv6b6)

## [Domain Model](#_8h2ehzxfam4o)

The Entity class is a parent class for the Game, Team, and Player classes. It provides each of its children with id and name properties as well as the getId, getname, and toString methods. The entity class also encapsulates the default constructor to ensure we cannot create a blank default class.

The Player class contains the constructor and its own toString method.

The Team class contains an array list of Players containing Player objects. It has its own constructor, an addPlayer method which uses the Iterator Pattern to add a new player to its array list and a toString method. The Team class also has a zero to many relationship with the Player class.

The Game class contains an array list of Team objects. It has its own constructor and an addTeam method which uses the Iterator Pattern to add a new team to the array list and it contains its own toString method. The Game class has a zero to many relationship with the Team class.

The GameService Class is a singleton so it only can have a single instance. It contains an array list of Game objects. It also has long fields: nextGameId, nextPlayerId, and nextTeamId. It has a zero to may relationship with the Game class.

**"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 server was low-cost popular server application. However, it has been discontinued. | Linux is widely used in server-side applications so there is a plethora of supporting material. It is also open source which makes costs lower. | While Windows server has robust toolset, it is not free. | A mobile device would not be a suitable agent for hosting a web application. They have low storage, low hardware performance (compared to server infrastructure), and are not designed for constant read/writes and running a reliable server. |
| **Client Side** | MacOS is only available for Mac hardware which can limit the scope of users. Because it is a Unix-based operating system, it is like Linux in structure which can make development easier. | Because Linux is open source, it is the lowest cost platform you can use. There are many distributions to choose from as well. The expertise level does need to be higher than the rest of the operating systems as Linux is very “plug and play” compared to other operating systems. | Windows has a large userbase and can be installed on a variety of hardware. However, it is a costly operating system in comparison to the open source Linux. | For mobile devices one would need to develop a web application or maintain a separate iOS and Android application with their respective stores. Mobile also must ensure that they are meeting Apple’s app store requirements which can be difficult and time-consuming. Android can be used on a wide variety of devices while iOS is only available on apple hardware. |
| **Development Tools** | MacOS has XCode which is useful for developing, debugging, and testing iOS applications. VSCode is also available which is highly extensible and can support a variety of languages, tools, and plugins. For programming languages: MacOS can support swift natively which is used for Apple operating systems. Electron is another excellent choice which is a JavaScript framework used for cross-platform native app development. Java is another cross-platform choice. | Linux can support VSCode as well as Sublime text for IDEs. Sublime text is a light weight and extensible IDE. VSCode has a much larger ecosystem and supports a variety of languages tools and plugins. We could use a variety of languages and frameworks on Linux such as Java, C++, or Electron. Java and Electron are designed to be cross platform and can support multiple operating systems. | Windows supports Visual Studio as well as VSCode. Visual Studio is capable of developing .NET applications. VSCode is a highly extensible and customizable IDE that can support a variety of languages, tools, and plugins. Windows can support a variety of languages and frameworks such as C#, Java, C++, and Electron. Java and Electron are built for cross-platform development. | The languages and frameworks that are used for mobile development would be Kotlin, Swift, and React Native. Kotlin is designed to interoperate with Java. Swift is used for iOS, iPadOS, macOS, and a variety of other apple systems. React Native is a React/JavaScript framework designed for cross-platform mobile applications. |

## 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**:  
     
   For a wide reach and great development experience, Windows is the recommended operating platform. Windows can be installed in a wide range of hardware such as desktops and laptops and supported by a multitude of hardware manufacturers. While Apple has a competitively large user base, MacOS is only utilized in an all-inclusive ecosystem between hardware and software which can be restrictive long term. While Linux has the advantage of being open-source and would be cost-effective, it is not as widely used or preinstalled in machines in the same manner as Windows, which would limit our user base.  
     
   From a developer experience perspective, Windows has a large variety of tooling that can be utilized. Windows supports all major IDEs such as Visual Studio, Visual Studio Code, Eclipse, Sublime Text, and others. This allows greater flexibility for developer choice and expertise for long-term hiring and support.
2. **Operating Systems Architectures**:   
     
   The Windows operating system is designed to be extensible, performant, and secure. Windows uses a layered architecture of modules which run on top of the kernel to ensure that applications are not directly communicating with the hardware. This helps ensure the system can be more secure from malicious programs while also making it easier for application developers since they do not need to write their own drivers.  
     
   Modern systems use a 64-bit CPU architecture which provides multi-core support. The benefit of this is the increase in concurrent tasks and the ability for the system to utilize more RAM for applications. By having more available RAM, the system can run additional tasks simultaneously or single applications can utilize additional memory in RAM instead of loading directly from the slower hard drive.
3. **Storage Management**:  
     
   Local user and game data can be stored on the hard drive of the Operating Platform. This would include source code as well as data fetched and synchronized with the server. Game data stored on a local machine would include statistics such as win/loss, user profile information, and meta data around each game. Upon load, the application will synchronize with the server to ensure that all information is up to date with the latest game data.  
     
   Because the application relies heavily on the anonymity of potential images, they will need to be stored on an Amazon S3 bucket and fetched during gameplay. This accomplishes two things. First, it allows the developers to constantly introduce fresh content to users to keep the game interesting and challenging. It also helps to ensure game integrity since users will not have direct access to image files and therefore could not look up images and chat like they would if images were all stored locally.
4. **Memory Management**:  
     
   One fantastic strategy that will be able to be utilized for memory management is dynamic loading. This will keep the routines on a disk and not load them until they are called. This ensures that unused routines are never loaded, which will help reduce the amount of memory overhead.  
     
   Windows will also allow the application developers to utilize swapping. This will use a backing store to temporarily swamp in and out of memory as needed. This ensures that we can optimize the faster processes while swapping the finished processes.
5. **Distributed Systems and Networks**:   
     
   Draw it or Lose It will utilize the Infrastructure as a Service of Amazon Web Services. Amazon Web Services will allow the application to scale as needed due to their pricing and packaging model. Because we will not be managing the physical hardware, we will be able to deploy services to different regions if necessary. Because we can deploy to multiple regions, this also will help to ensure that there can be a fallback region in the event of a server outage.  
     
   As stated in the Storage Management section, the Application will also utilize Amazon S3 for image storage. This is also a scalable solution as you will only pay for the storage space that is needed. It also provides the flexibility for the development team to update the image database from anywhere instead of a single location.
6. **Security**:   
     
   Security is a major factor in this application’s development. Because we will be utilizing local and distributed services to host and provide images and user data, it is important that the integrity of the communication between services is maintained.   
     
   First and foremost, there needs to be clear authentication for each game session and each user within the session. There are a variety of services that can serve as an authentication middleware to ensure each session is clean. Auth0 is a fantastic authentication service that would be decoupled from both the server side and client side of the application. During game sessions, each user would authenticate into the session through Auth0 and be provided a token to our data service to initiative the session. For security in our distributed systems, we will want to utilize Amazon Cloud Security. This is provided by Amazon with use the other Amazon Services.   
     
   Windows provides both Windows Defender and windows firewall with the system that is continuously updated. Windows Defender will use its database to regularly scan the system in search of malicious software such as viruses and trojan horses and remove them from the system.