

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 3.0

## 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 | 3-19-2023 | Jenna Case | Adding all information |
| 2.0 | 4-2-2023 | Jenna Case | Adding the Evaluation Section |
| 3.0 | 4-11-2023 | Jenna Case | Adding the Recommendation Section |

**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)

Creative Technology Solutions create Draw It or Lose It in a web based version of the gaming app. The purpose of the game is to have four rounds of play lasting one minute each. The rounds consist of drawing being rendered at a steady rate and are fully complete in 30 seconds. The team must guess the puzzle before time is up or the remaining teams have an opportunity to guess.

## Requirements

* A game will have the ability to have one or more teams involved.
* Each team will have multiple players assigned to it.
* 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.

## [Design Constraints](#_2et92p0)

1. A game will have the ability to have one or more teams involved.
2. Teams need to have multiple players.
3. Only one instance of the game can exist at any given time.
4. Game names and team names must be unique and must be checked at the start of every game.

## [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)

Each class is shown with the class name, public and private attributes, and any other operations. This uses encapsulation. Members and operations accessibility to other parts of the program can be controlled by making them private (-) or public (+). Also data members and operations are encapsulated in each class that they are designed for. Everything to be used within a class is inside it.

Inheritance is shown between the Entity, Game, Team, and Player classes. The Entity class is the base class for Game, Team, and Player. This means that they inherit the characteristics established in the Entity class.

Other association is shown between the GameService, Game, team, and Player classes. They are not subclasses of each other or inherited from each other so we use association to show they use the class for a purpose. For example, Game holds a list of Teams, Team holds a list of Players, and GamerService has a list of Games. The association shows zero to many, meaning that Game can have zero or more teams.

Polymorphism is shown with the toString() method of the Java base object class. It is used in the base class Object. The Entity, Game, Team, and Player classes override and implement the toString in the way that they need to use it. They all use it to print details specific to their class information.

Abstraction is used to hide a lot of the complexity in the data members and operations. In the Game, Team, and Player classes array lists are used to store their class objects. The Team class has an array list of the players can use this without worrying about the structure of the players list.

**"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** | Easy accessibility and server configurations. Has a simple graphical interface. There’s a lot of flexibility with frequent OS upgrades. Can integrate easily with iPhones. | Free to use. The platform is difficult to navigate. Many distribution choices and is opensource.  Some providers may prefer Linux to other operating systems. | Windows is widely used. Lots of hardware options but can be inconsistent with functionality between versions.  License costs are higher as they are by user. | Specifications differ from user to user. Specifications are better in other devices. Cost is a big variable because it would need to be built in house.  Hardware on mobile devices is more limited. |
| **Client Side** | Expensive for users. More skills are needed to navigate the operating system.  Mac has a smaller customer base as opposed to Windows so opportunity is smaller. | A lot of expertise is required to operate. Linux data is required to use the operating system. | Easy to learn how to use and support a Windows setup. Less experience needed overall on the client side.  Windows is far more preferred for users than other OS so it has a larger opportunity. | More difficult to implement than on other devices. Better flexibility for updates. Mobile devices are not designed as multi-user.  Developing for Android and iOS could prove to be difficult. |
| **Development Tools** | Mac uses C and SWIFT for development languages.  Would need to use Xcode as the IDE. Xcode is listed at $99 per year. | Linux uses C/C++, Java, Python, and Perl.  Could use PyCharm IDE for Python. Eclipse could run the IDE for C/C++ and Java on Linux. Both PyCharm and Eclipse are free to use. | Windows uses C/C++/C# and .NET as the primary development languages.  Microsoft Visual Studio would be the IDE to use and can code in many languages. Microsoft’s Visual Studio costs $499 per year. | Android and iOS development languages will differ. Android primarily uses Java. iOS uses C and SWIFT.  For Android, Android Studio by Google would be the IDE which is free.  For iOS, Xcode would be used same as Mac. Xcode costs $99 per year. |

## 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**: I would recommend that they use Windows as an operating platform. Since it is already run in an Android app, it is developed in Java which can be run on Windows as well. Windows is also familiar to many people anyway, so development becomes easier.
2. **Operating Systems Architectures**: Windows is used by many people already so it makes it an easy OS for clients. It is developed and published by Microsoft. Windows also has better tooling like IDEs and debuggers making it good for development. Windows also allows users to develop the environment exactly the way they want it. It allows for easy transfer if the need arises to test different applications on a different OS as well. A backend server would manage the game environment and the frontend would be for rendering.
3. **Storage Management**: Windows has Storage Spaces which protects data from drive failures and can extend storage over time as you continue to add drives to the PC. It is possible to create an SQL database through Microsoft, which is another way to store information. On the development side I would recommend a hard drive for storage. On the client end, I think a cloud based storage application would be best.
4. **Memory Management**: Storage Sense would allow for storage and memory management. This allows everything to be kept together in one secure spot in the memory. Windows allows the developer to specify the amount of memory needed for the application to run as designed. This way we can ensure that the game will function correctly, since we can allocate the correct amount of memory.
5. **Distributed Systems and Networks**: Network games usually have a database shared among the players that are distributed and can interact with each other over the network. CTS would need to implement the shared database and the inter-player communications so that it can distribute to the app.
6. **Security**: Windows has security built in. As a developer, the Security Compliance Toolkit can be downloaded which can manage security baselines. Windows lets the user design firewalls to protect information as well as using encryption to keep data safe.

To build up security, they can also use role-based authorization. I would recommend they limit the scope based on the type of user in a hierarchy. There could be a guest player and a team captain player in order to create separation in gameplay. None of the users would have admin access so only The Gaming Room can make those changes.