

<Draw it or Lose it>

# **CS 230 Project Software Design Template**

Version 1.2

## 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 | <11/12/2022> | <Steven Anderson> | This revision includes completion of the player and team classes and applying an iterator pattern to game, team and player. Also include adding and running the Entity class as a base class for game team and player inherits from. |
| 1.1 | <11/27/2022> | <Steven Anderson> | Revised and revisited document in preparation for resubmission. |
| 1.2 | <12/2022> | <Steven Anderson> | Revised with my recommendation section being filled in. |

## [Executive Summary](#_sbfa50wo7nsh)

The client “The Gaming Room” requests a web-based game that operates on more platforms than its current android. Their game is named “Draw It or Lose It”. The game involves random images from stock drawing that the teams have to guess what it is before the 30 second timer expires otherwise the other team gets 15 seconds to guess. “The Gaming Room” doesn’t recognize how to set up the environment for the web-based version.

## [Design Constraints](#_2et92p0)

* Will be run on different platforms
* Will have one or more teams
* Teams will have more than one player
* Only one game or team of a given name will exists at a time and the code needs to check to see if that name is already in use.
* Only one game instance should exist at a time.

## [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)

Program Driver is used to run SingletonTester in its main(). Information from Entity will be inherited by Game, Team, and Player classes. These classes will all have the id and name in the superclass Entity. Also using aggregation, GameService has a list of games from Game and Game has a list of teams from Team and Team has a list of players from Player. GameService is where new games checked to see if they exist. Games is where new teams are checked to see if they exist. And Team is where new players are checked to see if they exist

**"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.**

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | * Terminal commands are easy for configuring * Upgradable options * Popular for web hosting * Not normally preferred for web hosting | * Cheaper * Easier to find security weaknesses * Very modular in developing, allows for more custom choices * Most preferred for web-hosting * Takes more work to get it to work with hosting * Getting software can be tedious and might be filled with viruses | * Most user-friendly option * Has the most available software to choose from * Comes with control panel * Not as secure as Linux * Expensive * Additional software used can add to the price | * Popular platform * Cost- effective * Portable * Less secure than other options * Varies with different smart phones * Servers work better when stationary |
| **Client Side** | As long as the client is familiar with Mac there shouldn’t be much time required. Will take longer to see if it is compatible with other platforms. | Will require the most training to get client comfortable with use. Though will be cheaper than other options. | Least amount of time for training since it is the most widely used OS. And it is about as expensive as going with the Mac option. | Allows client to check status from anywhere. Though will require more time for client to make work with other platforms. |
| **Development Tools** | XCode is a good software for this on Macs.  Can run every language it needs to. Which for this should be languages like HTML or CSS or JavaScript. Swift is a good language for the desktop app. | Linux can run Visual studio and Eclipse for this.  Can also run the languages needed like HTML, CSS, and JavaScript. C languages are good for the desktop app. | Runs similar to Linux by using Visual Studio or Eclipse. Will also use the same languages like HTML, CSS, and JavaScript. C languages are good for the desktop app. | With Apple products you can use XCode like a Mac. With Android you can use AWD (android web developer) or AIDE web. With both languages options you can run languages like HTML, CSS and JavaScript. Swift would be used for Apple while Java works well for android with the app |

## Recommendations

1. **Operating Platform**: My recommendation would be for The Gaming Room to use Windows since it requires less time to train with and more available software to help them set up. Also, it will let them use IDE’s they will be familiar with.
2. **Operating Systems Architectures**: Using Windows allows them to take advantage of Windows services that connect all windows applications. These services will help them access the system resources and provides a user interface for assistance.
3. **Storage Management**: Can always use cloud-based storage for a more secure storage but if you want to use a physical storage space you can use the windows storage system that allows for easy creation and storage of files onto a computer. Additionally, with how freely you can look at and modify system files you can manage those files fairly easily on Windows.
4. **Memory Management**: With Windows you can store the photos needed for the game easily in folders with the game software so that your database of photos can easily be kept together.
5. **Distributed Systems and Networks**: You can use programs like develop 4 to help create cross-platform games. Once a game is made the files can be exported into a web-based form. If the servers are strong enough and have proper security the user base should be manageable by the server.
6. **Security**: Windows has included security software for protection along with plenty of 3rd party options to keep it secure. While this may add to the price, I think it is worth it to use the most used system. Its security software scans frequently and can provide real time updates and threat protection. Regular updates from Windows keeps their built-in security software up to date and with 3rd party software you got options for even more specific protection or more intensive protection.