

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 | 01/16/2022 | Joshua Fallin | Established executive summary, design constraints, and domain model |
| 1.1 | 02/06/2022 | Joshua Fallin | Established Development Requirements within Evaluation |
| 1.2 | 02/20/2022 | Joshua Fallin | Established Recommendations |

**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 created a mobile game called “Draw It or Lose It” available strictly on the Android market. The plan is to port their game to other platforms, such as Mac, Windows, Linux, or other mobile devices. Before this can be accomplished, we must first decide which platform to invest in. All three platforms have various strengths and weaknesses that will be outlined. It is important to focus on the infrastructure that each platform provides, as well as the development tools and programming languages. All three options have stark differences that must be addressed.

Considering these aspects, it is our recommendation to focus on one platform for now. This will allow us to utilize our resources reasonably, and possibly make porting over to another platform much easier, giving us a template to follow later.

## [Design Constraints](#_2et92p0)

* System constraints
  + One only one game can exist in memory, requiring unique identifiers for each game, such as an ID and name
  + Games can consist of one or more teams, then each team can have multiple players. This implies a one-to-many ratio between game and team, and team to players.
  + Each team and player must have a unique name, so that they can be identified
  + Large database of images is required. Beneficial to start with the current stock of photos available on the Android version. This can be done using their database.
* Technical Constraints
  + Application must follow a web-based model for later implementation with other platforms.
  + The dominant language should be based on the current language the Android app, currently Java. This will allow for easier communication between the systems.
  + The current cloud service must be reanalyzed and either ended or renegotiated. The potential increase in users will mean a greater strain on our resources and could incur overage fees or hamper server performance.
* Business Constraints
  + Regular milestones will be imposed. A schedule will help to control costs, direct workflow, and provide regular updates to the client.
  + Using the stock photos on the current Android version will reduce the budget, since we will not have to pay for copywrites for more images the players draw.
  + More images can be purchased later, to attract current users to the new platform. Giving them a unique experience and encouraging them to invite other users.
  + Hosting services will continue to be outsourced, valuing design of the platform over time spent on in house hardware and management of the server.

## [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)

Starting with the “Entity” class, you can see it has two attributes, a unique id and name. The entity has several methods and two constructors. The default constructor is kept private, while the overloaded takes two parameters, which sets the id and name. Two accessor methods are needed, getId() and getName(). Lastly, is the override toString() method. There are three subclasses of the Entity class: Game, Team, and Player. All three inherit from the Entity class.

The player class manages the “players” and has a constructor that takes two parameters, id and name. The second method is a toString() override. The team class consists of a list of player objects and includes a constructor and two methods. The first constructor takes two parameters, id and name. The other two methods are addPlayer(), which adds a player to the player list, and a toString() override.

The game class is a private attribute made up of a list of team objects. It includes a public constructor which takes two parameters, id and name. The other two methods are addTeam(), which adds a team to the list, and a toString() override.

The GameService class is based on a singleton instance template. It has various attributes including a list of games, three long type variables called nextGameId, nextPlayerId, and nextTeamId, and a service variable called GameService to support the singleton instance. There are subsequent methods and constructors such as the private constructor GamesService() to support singleton. Others include get Instance() which returns the GameService instance, addGame() which adds a new game, and accessor methods for game, game count, player, and team.

There are several relations between the various classes. As already pointed out, the game, team and player classes inherit from the Entity class. Teams can be made up of zero to many players, Games can be made up of zero to many teams, and a game service can have zero to many games.

The program Driver contains the main() method and uses the SingletonTester class. The SingletonTest class contains one public method, testSingleton(). This tests for a single instance of the Game Service object.

**"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 provided its own programming language for server-side support called Swift. This language also provides support for Linux. One of the great strengths of a Mac is that both Mac Servers and clients communicate seamlessly. Communication and setup are very simple when linking to two. | Linux is an open-sourced OS which helps reduce some of the licensing barriers one experiences with Windows and Mac. They also allow for great customization. However, there is no guarantee of seamless communication across a range of various Linux devices. | Windows is a popular and powerful OS. They provide a wide range of resources for managing the server side of applications. Of key importance is their Azure web service, designed to work seamlessly with Windows. | Mobile devices are a broad based term and the most popular divide into two categories, Android and Apple. Apple is based on the MacOS, already outlined in another column. Android is Java based and there are various options for server hosting such as Amazon’s AWS and Microsoft’s Azure. Both provide sound webhosting resources |
| **Client Side** | Development for multiple Mac devices is the same as developing for one Mac device. Some key things to look for are developers skilled in Objective C, Swift. An initial investment is expected to be between $150,000 - $200,000 and can take as little as 4.5 weeks to up to 12 weeks for a simple version of the app. | Developing for Linux devices should include developers skilled in various programming languages, predominantly C and/or Java.  As with Mac, an initial investment is expected to be between $150,000 - $200,000 and can take as little as 4.5 weeks to up to 12 weeks for a simple version of the app. | Developing for Windows based devices should include developers with skills in C++. As with the other two OS’, an initial investment is expected to be between $150,000 - $200,000 and can take as little as 4.5 weeks to up to 12 weeks for a simple version of the app. | Client based needs for mobile devices should include a team of developers experienced with Android *and* Mac devices. This includes Java and Objective C. It should be noted that with a mobile version, the app should work on both platforms to reach the most people. |
| **Development Tools** | The relevant languages of iOS is Swift and Objective C, with Swift becoming the dominant language. Important tools include XCode, an Apple supported IDE | The relevant languages for Linux include C and Assembly. Any IDE that supports C or Assembly could suffice for development. Linux does provide a powerful SDK within the GNU toolchain | The most relevant languages for Windows is C, C++, and C#. Important tools include Visual Studio and the all encompassing SDK kit provided by Microsoft called Windows App SDK. | Mobile development languages vary based on the OS. For Apple phones the same tools used to develop iOS programs is consistent, Swift and Objective C, as well XCode. For Android phones Java and Kotlin are the dominant languages. Important IDE tools include Android Studio, an SDK provided by Google. |

## 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**: Azure Cloud Service

An Azure cloud service allows for scalability while keeping maintenance cost manageable. One of the benefits of an Azure service is it gives the client (The Gaming Room) the ability to focus on development. The service comes with powerful tools for controlling bandwidth and providing live updates to the client software with little down time. Negotiating the cost of an Azure service will be important. While setting up our own server has high startup costs, maintaining a cloud service incurs long term costs. However, as a startup app “Draw It or Lose It” can monitor its growth using the built in Azure monitoring tools. If bandwidth costs get too high, we can take our intellectual property and move it to our own server in the future.

1. **Operating Systems Architectures**:

Azure cloud services provide a suite of virtual machine hardware. It allows us to customize our needs based on growth and user patterns without having to manage physical hardware. It is a combination of Software As a Service (SAAS) and PAAS (platform), allowing us to outsource the hardware and focus on the application.

1. **Storage Management**:

Azure storage is virtualized. It allows us to expand as needed. One of the key features of this is that since we are keeping the images on the server, we can define how much space is needed, thereby controlling costs. Another important aspect is that Azure supports database management. This can be used to help us manage user login credentials and provide higher security.

1. **Memory Management**:

One of the benefits of Azure is that it supports memory ballooning. This helps to better manage freed memory when a virtual machine no longer uses it. This can ensure that when multiple games are running, they are given adequate memory to operate on the server.

1. **Distributed Systems and Networks**:

Using an Azure cloud service we can base the network connections on an HTTP connection, similar to a Dropwizard application. With an Azure VM we will be able to streamline connections between Mac, Linux, and Windows based OS.

1. **Security**:

Azure has many built in security features. For example, Microsoft Sentinel which provides security and event management. Azure resource manager allows you to manage directly who can do what with the VM. Azure Monitor also allows administrators to observe and monitor usage on the server VM. This provides the client with tools to monitor potential security concerns.