

<Draw It or Lose It>

# **CS 230 Project Three**

Version 3.0

## 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)5

[**Recommendations**](#_m8aleynsvzvc)5

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | <03/27/22> | <Noah Coleman> | <Initial Design Document> |
| 2.0 | <04/03/22> | <Noah Coleman> | <Updated Evaluation> |
| 3.0 | <04/17/22> | <Noah Coleman> | <Updated Recommendations> |

## [Executive Summary](#_sbfa50wo7nsh)

This project will focus on developing a web-based game for The Gaming Room that serves multiple platforms. It will be based on the client’s current game, Draw It or Lose It, currently only available on Android. The game will access stock drawings, and slowly render them over 30 seconds. The current team must guess the drawing, or hand control to the opposing team.

## [Design Constraints](#_2et92p0)

1. Cross platform functionality

The client’s main requirement is that the app be available on multiple platforms. This constraint will require the team to have knowledge of multiple operating systems and be able to implement the application on each.

1. Image Library Access

The app will function by accessing a library of stock images. We must ensure that access to this library does not become interrupted, and that we have the rights to the images.

1. Web based play

Users will play online against other teams in a live format. Our app must support multiple teams logging in at once. We must make sure that each team is unique as well as each game being unique.

## [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)

The UML diagram shows how we intend to execute this project. First, we will create an Entity class which will be the base class for the entire program. All other classes, except our driver and our singleton tester, will inherit form the Entity class. The remaining classes will all share an association but will only inherit from the Entity class.

The GameService class will create the initial instance of each game, creating a gameId, playerId, and teamId. This class has a 0 zero to many multiplicities with the Game class, meaning that there could be no instances of game, or there could be many. The Game class creates a teams list, which then has a 0 zero to many multiplicities with the Team class. The Team class creates the players list, which then has a 0 zero to many multiplicities with the Player class.

The Driver class uses the SingletonTester to ensure that only one instance of GameService can exists at one time. Doing this makes sure that each game instance is unique.

**"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** | * Dedicated Mac OS X server could host the application * Mac is a widely used operating system, making it easy to implement. * Ability of Mac to link up with apple’s mobile platform would make cross-functionality ideal. | * Has supporting software not exclusive to Linux * “Has symmetric multiprocessing which allows separate processes to execute in parallel on separate processors,” (Wiley, 21.5.3) * Supports protocols regularly found on large PC networks * Linux is entirely open-source, which might create security risks. * Not the most widely used, so might need a specialist to implement. | * Windows uses a client-server architecture, and processes user level subsystems * “Designed to provide high performance on desktop systems, server systems, and large multithreaded… environments,”(Wiley, 22.2.4). * Widely used and familiar among programmers. | * Mobile applications have low server capacity compared to traditional operating systems. * Cost would be lower, but so would speed of execution. |
| **Client Side** | * Server cost is low compared to Windows. * Time and expertise costs should be low considering Windows prevalence in the marketplace | * Low cost due to open-source nature of Linux * Need someone proficient in Linux to implement the operating system, costing time and expertise. * Each call to an execution of a user program runs a new program and overwrites the current execution. Might cause delays. | * Windows would present a higher startup coast to purchase space on their server. * Time and expertise costs should be low considering Windows prevalence in the marketplace. * Client-server architecture accounts for ease of execution in client-side server calls. | * Mobile development is less expensive than traditional development. * Time and expertise costs should be low, there is a growing number of developers with mobile development experience. * Android and IOS have their own development platforms. |
| **Development Tools** | * Objective-C is the most used language for apple products * Swift is Apple’s open-source language * HTML * C Language | * C Language is commonly used to write system applications integrated into Linux * Visual Studio and Eclipse are two IDE options for Linux | * C# helps create XML web services for Windows and specializes in Windows-based platforms * C Language is commonly used to write system applications integrated into Windows * Microsoft has their own IDE, Visual Studio | * Swift is the language used by IOS developers * Java is the programming language for android 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 this application I am recommending that we develop on Windows. Windows is one of the most common operating systems in use, which will make it the most adaptable to other computing environments. Windows contains a selection of UI frameworks to chose from, which will make the app more seamless for the user. Additionally, Windows storage structure will allow us the storage and memory usage necessary for a clean application.
2. **Operating Systems Architectures**: Windows architecture is based around a layered design consisting of a user mode and a kernel mode. The user mode only has access to limited memory, while the kernel mode is unrestricted. This would be appropriate for our application as the user only needs access to the functionality of the game. The kernel mode will give developers the ability to access all system resources necessary to create and make updates to the game.
3. **Storage Management**: Windows uses a hybrid file services storage structure to increase hosting capabilities. The hybrid model consists of on-premises file servers as well as Azure cloud-based file servers. This hybrid model will allow the application to access user accounts, photos, and game data wherever the application is being used. The Storage Sync service utilized in this design creates trust between serves and resources.
4. **Memory Management**: Memory management is built into Windows using Windows Virtual Memory Manager. “Windows SQL Server dynamically acquires and frees memory as required,” (LitKnd). Utilizing Windows built in memory management tools the application would be able to draw pictures from storage and place them into memory for use by the game. Reduced disk I/O means the application would run more quickly, providing a better experience for the user.
5. **Distributed Systems and Networks**: For this application we would use a client-server distributed network. Each client application would have to connect to the server hosting the basic function and storage for the application. The client applications would have to be written to draw off the server simultaneously for play across platforms. A server network would be required to allow users to connect to the server and draw from the same pool of photos and access the same game platform.
6. **Security**: Windows bases their security round user accounts. This allows system admins to all or deny permission to system objects. Users are only allowed access to a certain area of the system and nothing greater. The application can be seen as a user in this system. Each instance of the application will be able to gain access to the server to retrieve account information and pictures for the game. They will not, however, be able to modify server level code or access user accounts outside of the scop of the current user logged in to the application.

References

Arslan, A., & Ali Arslan (82 Articles Published) . (2021, December 23). *The 10 best ides and code editors for linux*. MUO. Retrieved April 1, 2022, from https://www.makeuseof.com/best-ide-code-editors-for-linux/

*Guide to programming languages*. Code a New Career | ComputerScience.org. (2022, March 18). Retrieved April 1, 2022, from https://www.computerscience.org/resources/computer-programming-languages/

LitKnd. (n.d.). *Memory Management Architecture Guide - SQL Server*. SQL Server | Microsoft Docs. Retrieved April 17, 2022, from https://docs.microsoft.com/en-us/sql/relational-databases/memory-management-architecture-guide?view=sql-server-ver15

Wikimedia Foundation. (2022, February 18). *Architecture of windows NT*. Wikipedia. Retrieved April 17, 2022, from https://en.wikipedia.org/wiki/Architecture\_of\_Windows\_NT#Kernel\_mode

Wiley. (n.d.). *Operating System Concepts, 8th edition*. O'Reilly Online Learning. Retrieved April 1, 2022, from https://learning.oreilly.com/library/view/operating-system-concepts/9780470128725/silb\_9780470128725\_oeb\_c08\_r1.html?sso\_link=yes&sso\_link\_from=SNHU