

Module 3

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[CS 230 Project Software Design Template](#bookmark)1

[Table of Contents](#bookmark1)2

[Document Revision History](#bookmark2)2

[Executive Summary](#bookmark3)3

[Design Constraints](#bookmark4)3

[System Architecture View](#bookmark5)3

[Domain Model](#bookmark6)3

[Evaluation](#bookmark7)3

[Recommendations](#bookmark8)5

## [D](#bookmark9)[ocument Revision History](#bookmark9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 03/19/22 | Branden McCusker | Created file |
| 1.1 | 04/01/22 | Branden McCusker | Updated the “Evaluation” portion of the file. |
| 1.2 | 04/16/22 | Branden McCusker | Updated the “Recommendations” portion of the file. |

**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.

## [E](#bookmark10)[xecutive Summary](#bookmark10)

Create a program game similar to the 1980s television game Win, Lose or Draw.

## [Design Constraints](#bookmark11)

Submitting this program under a web based format could be difficult directly due to the format.

## [System Architecture View](#bookmark12)

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.

## [D](#bookmark13)[omain Model](#bookmark13)

ProgramDriver is the driver for the code, having only main(). This calls SingletonTester, which only allows one instance to be ran at a time. Entity, GameService, Game, Team, and Player all contribute towards the functionality of the program. GameService manages the logic and math of the program and utilizes Game. Game, Team, and Player all implement Entity and manages their respective roles in the game.

**"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.**

## [E](#bookmark14)[valuation](#bookmark14)

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** | Very good for high level servers with ease of use and support. Most expensive option and little after-market upgrade capabilities.  Mac has built in optimization for servers.  The licensing costs are none, as the OS is free. | Very good for any level servers. Cheapest option.  Least user friendly.  Linux is very optimized for servers from install, and has the most control over server console.  The licensing costs are none, as the OS is free (potentially). | Middle of Mac and Linux in ease of use, cost, and support.  Windows has the most support console-wise, and is decently optimized.  The licensing costs are the most, as Windows costs money. | May be the cheapest way to run a server, but very limited support and resources.  The licensing costs are none, as the OS is free (potentially). |
| **Client Side** | Very user friendly.  Stable browser is built in.  Cost is potentially the highest. | Requires the most computer knowledge, but minimal cost and great for web development. | Middle of Mac and Linux in cost and ease of use. Windows may need to be purchased. | Potentially the cheapest and more than capable of running browser games. Very user friendly. |
| **Development Tools** | Mac, especially ARM-based produced, typically compile programs the fastest, and built in support for the latest Java and Python. Native support for Unix.  Apple produces the language Swift and SwiftUI, which is free to publish under. This is true as well with the IDE Xcode. | Potentially the best for programming, but requires OS knowledge and additional downloads besides an IDE.  Linux may be the fastest to compile a program. | Middle of Mac and Linux with compile times and price.  Microsoft produces the language Visual Basic, which is free to publish under. This is true as well with the IDE Visual Studio. | Slowest compile times, and requires external hardware for programming to be feasible (i.e. keyboard).  IDEs are available for download, but language support is minimal and often slow to update. |

## 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**: Windows has more support than Mac or Linux.
2. **Operating Systems Architectures**: Windows supports x64 and x86, as well as other outdated architectures such as x16 and DOS.
3. **Storage Management**: A PC could handle this, but a server is more realistic.
4. **Memory Management**: Storing both game data and user data into RAM while the program is running.
5. **Distributed Systems and Networks**: A single player mode and an online disconnect buffer is recommended. This would allow the browser to prevent a forced reload on an internet disconnect. In the case of multiplayer or a mode that requires data retrieval from a server, cookies and constant data uploads are recommended to resume play when the internet connection returns.
6. **Security**: Encryption and firewall support with automatic updates from Microsoft are required for long lasting security. These features are built into both Windows and HTML.