

CS230 Project One Game App

# **CS 230 Project Software Design Template**

Version 1.0

## 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 | 03/23/25 | Derek Castro | Modified cover page, provided executive summary on this week’s module, added new requirements, design constraints, explained the provided UML diagram, evaluated the various OS characteristics, and added recommendations for each platform. |
| 1.0 | 04/06/2025 | Derek Castro | Provided written evaluation tweaks on the OS characteristics outlined within the table included to emphasize and review the technical requirements and potential licensing costs to proceed with project development. |
| 1.0 | 04/20/2025 | Derek Castro | Reviewed recommendation OS with specified parameters including systems architectures, storage management, memory management, distributed systems and networks, and security. |

**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 is requesting assistance to help develop their web-based game based on the 1980s television game *Win, Lose, or Draw* where teams are competing to guess what image is being drawn. This module will focus on specific requirements as follows:

* A game will have the ability to have one or more teams.
* Each team may have multiple players.
* Game and team names must be unique to allow existing profiles.
* Only one game instance can exist at a time.

## Requirements

* Business Requirements
  + Obtain a detailed inventory of stock images to choose from.
* Technical Requirements
  + A game will have the ability to have one or more teams.
    - Implementation of iterator pattern for methods such as addTeam().
  + Each team may have multiple players.
    - Implementation of iterator pattern for methods such as addPlayer().
  + Game and team names must be unique to allow existing profiles.
    - Implementation of inheritance through creating a base class Entity to hold common attribute and behaviors.
  + Only one game instance can exist at a time.
    - Implementation using singleton pattern in the GameService class.

## [Design Constraints](#_2et92p0)

* The application must handle multiple games, teams, and players simultaneously.
* Each game, team, and player should have a unique name to prevent duplication which can cause confusion and overlap.
* Only one instance of the game can exist at a time to ensure that game performance is optimized.

## [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)

Class Relationships:

* The Game, Team, and Player classes inherit from the Entity class which contains common methods (getID(), getName(), toString()), and common attritbues (id and name).
* The ProgramDriver class uses the SingletonTester class to test the GameService class.
* The GameService class manages the list of Game objects which manages the list of Team objects which manages the list of Player objects.

OOP Principles:

* Encapsulation
  + The attributes are controlled through public methods (getId(), and getName()) which stores their data in a private method to ensure uniqueness of new names.
* Inheritance
  + The Entity class allows each object class (Game, Team, and Player) to utilize re-usable code to obtain the common attributes which can help prevent code repetition.
* Polymorphism
  + toString() method allows flexibility for the object classes to utilize methods differently based on which object type is called.
* Abstraction
  + The Entity class abstracts the common properties and behaviors which simplifies the code to allow the object classes to function easier.

**"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.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Usually more costly, but is reliable with a reputable graphic interface. | Great benefits for cost due to its open source, and free to use however this may result in lesser quality compared to other paid services. | Most common operating system which is usually more multi-platform functional but may be costly. | Not commonly used for web hosting which may cause many issues due to unfamiliarity and minimal available resources. Contains hardware limitations and lack of server-grade software. |
| **Client Side** | Mac usually has user-friendly and high quality services, which can reduce necessary learning curve time investing, but can be costly. | Provides great cost benefits with its free to use services but can require higher technical expertise compared to the other OS. | Offers great compatibility and user-friendly services but can be costly due to licensing costs. | May require specific development and can be difficult for compatibility with web-hosting such as Android Studio or Xcode. |
| **Development Tools** | IDEs such as VSCode and Xcode.  Language tools can include CSS, HTLML, and JavaScript. | Language tools can include CSS, HTLML, and JavaScript. | Language tools can include CSS, HTLML, and JavaScript.  OS specific can include, Eclipse, Pycharm, and Visual Studio. | Language tools can include CSS, HTLML, and JavaScript.  Specific IDEs for mobile can include PHP and Python. |

## 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**: The optimal OS that will provide a broad source of IDEs to choose from is Windows due to its popularity and multi-platform compatibility. Windows uses a ‘monolithic kernel’ (Linux) that handles core functions efficiently.
2. **Operating Systems Architectures**: Utilizes a kernel that handles core system functions which handles memory management, process scheduling, IO management, etc. which allows efficient resource management.
3. **Storage Management**: Although it may not be the most optimized strategy to manage web-based storage but having an external hard drive to backup important and large data files helps ensure data security to defend against any data loss event. For live game data, cloud storage can scale with player demand through Windows server integration.
4. **Memory Management**: Most modern web browsers provide automatic memory management to handle the game’s simultaneous resource utilization.
5. **Distributed Systems and Networks**: Implementing a cloud infrastructure can allow the game application to be distributed on multiple platforms to allow cross-platform compatibility and auto-scaling. REST Apis between servers and involved devices can ensure compatibility.
6. **Security**: Window’s built in Defender automatically helps protect user data between platforms. Implementation of user accounts using username and passwords can further improve user information. HTTPS encryption plays a crucial role with data transferring.