

The Gaming Room

# **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 | 11/16/2024 | Zachary Abbe | Changes to cover page, executive summary, design constraints, system architecture view, domain model and recommendation |
| 1.1 | 11/30/2024 | Zachary Abbe | Changes to the Evaluations table |
| 1.2 | 12/13/2024 | Zachary Abbe | Changes to the 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 project is to develop a web-based game that serves multiple platforms based on the

current game draw it or lose it, which is only available on android. The purpose of the game is to have

multiple teams consisting of several people going four rounds at a minute each. When a picture is pulled

from a library of images one team guesses till time runs out. If not answered each opposing team

member gets to answer till 15 seconds runs out.

## [Design Constraints](#_2et92p0)

* Only one instance of the game will exist at any time.
* Game and Team names must be unique to allow users to check whether a name already exists.
* Each Team should have multiple players.
* Must run on multiple platforms.

## [Domain Model](#_8h2ehzxfam4o)

Entity creates a relationship between Game, Team, and Player class. This means they all inherit from Entity. With UML we can show this with inheritance. Making Entity a superclass. When

we look at their relationship, Team and Player is a “has a” type. While Game has a Team and

GameService has Games. When using UML, it’s called aggregation. When a user “has a” I mean

it's an instance of one class and has a reference to an instance to another class. When we look at this

diagram, GameService has a reference of Games, Games a reference of Teams, and Team a

reference of Players

**"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** | -Easy to use GUI  -Comprehensive terminal commands  -Easy server Configurability | -Very cost friendly  -difficult GUI  -many commands lines for different things  -easy to run servers | -Easy to use GUI  -uses command prompt for terminal commands  -very expensive to maintain servers | -Specs dependent on price  -easy to use GUIs  -Massive portability |
| **Client Side** | -Very expensive to jump in  -expertise required to program  -Can be a bit difficult to manage macOS | -massive expertise required  -intensive knowledge to run the program in linux  -cheapest of the platforms to get into | -little expertise is needed  -pricepoint middle ground with windows  -Easy to learn, and support a windows platform | -more difficult to implement than the most devices  - unique client side integration necessary |
| **Development Tools** | -Languages: HTML, JavaScript, and CSS.  -Vast libraries  -selection of dev tools to use on macOS: GitHub, Visual Studios, PyCharm | -Languages: HTML, JavaScript, and CSS.  - vast libraries  -Linux supports JavaScript, Ruby, Python, and PHP. | -Languages: HTML, JavaScript, and CSS.  -vast libraries  -Dev tools: Visual Studios, Eclipse,  CMD Prompt, PyCharm. | -Languages: HTML, JavaScript, and CSS.  -Vast libraries  - Multiple IDEs for languages for HTML, PHP, C++, C#, 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**: Overall, I actually choose to use a web based Linux server for hosting the Draw it or Lose it game. I choose Linux specifically for their low cost and their very high level of security. We can use Amazon Web Services to use their cloud computing, that will take longer to deploy and configure properly, but overall will be our best option. Amazon Web Services virtual server instances are highly secure and will help us prevent bad actors from accessing things we don’t want them too. Also using a web-based cloud computing solution allows us to ramp up deployments in order to meet the demand needs of our users.
2. **Operating Systems Architectures**: Linux OS is a modular operating system that allows for maximum customization and can include a wide variety of security services, and business logic solutions. Using cloud servers allows us to only pay for what we need to use and doesn’t include a lot of other bloat features that wouldn’t be useful in the deployment of the game. Overall, it will take less storage and still allow us to fully utilize its potential.
3. **Storage Management**: I believe that Amazon Web Services will be best due to cost effectiveness. This eliminates our need to purchase and maintain our own hosting equipment and allows us to cut down on the necessary staffing costs that would arise from having our own server team. This also can allow us to manage backups more efficiently. If we Use Amazon Simple Storage Services, we can then manage our cloud storage needs. With that in mind, we can also use something like PostgreSQL for our database in the game because it’s an open-source solution.
4. **Memory Management**: To manage memory the best we can do several different things:

Using the appropriate format for our images used in game, like JPG, will allow for a smaller file size to store on our server. Sizing the game images to their container size, so that there’s little overlap between the sizes of different images. We can utilize image compression to ensure that we can get a smaller size, without losing quality and using the correct network requests in order to cache incoming images so they load faster.

1. **Distributed Systems and Networks**: We will be using PostgreSQL databases, Linux-based virtual server instances, and the clients of our users for our distributed systems and networks. All of this can be managed via the cloud, so that we can increase capacity dynamically. This allows us to create more servers when the user demand is higher than we currently have. This also lets us mitigate outages, as we can deploy servers in many different locations, with minimal impact to the user. For our networking we can use a variety of tools from AWS. This includes AWS direct connect, AWS direct connect + VPN, and AWS VPN cloud hub. This allows traffic to be distributed across a variety of networks, instead of a single pipeline that can slow down with many requests. This helps fight lag for our userbase.
2. **Security**: Since we are going to be using a Linux based deployment, we can automate security tasks thanks to using AWS. Strict Authentication and Authorization will be necessary for anyone trying to manage the tasks in the stack. Using role-based access also prevents users from accessing key/ critical infrastructure of the game and injecting other code to execute. This also can allow just certain members of the users’ team to change any game settings for their match. Overall the security is also enhanced based on the devices used by the user. For example iOS uses Data Protection, that encrypts/decrypts files in the app directory. Android uses a variety of different Anti-Virus/ Encryption protocols based on the manufacturer of the device, and Windows can also use a variety of these tools. This allows us to handle the security for the games integrity on our end, and the users end.