

Draw or Lose it

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

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

[**Table of Contents 2**](#_30j0zll)

[**Document Revision History 2**](#_3znysh7)

[**Executive Summary 3**](#_2et92p0)

[**Requirements 3**](#_tyjcwt)

[**Design Constraints 3**](#_1t3h5sf)

[**System Architecture View 3**](#_4d34og8)

[**Domain Model 3**](#_2s8eyo1)

[**Evaluation 4**](#_17dp8vu)

[**Recommendations 5**](#_26in1rg)

## [Document Revision History](#_lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | <09/12/2023> | <CJ Cline> | Win it or Lose it |

**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](#_35nkun2)

The goal is to make a web based game of draw it or lose it, web-based distribution method will be over web browsers. Your challenges will be primarily internet based, you’ll need to be able to provide several different access points and a way for people to directly interact with the specific game they are looking for. A strong example of how this could work would be looking at the JackBox game model. Where there is a host machine and people connect to it via a room code provided by your website but the game process is handled elsewhere.

## Requirements

*A game will have the ability to have one or more teams involved.*

* *Each team will have multiple players assigned to it.*
* *Game and team names must be unique to allow users to check whether a name is in use when choosing a team name.*
* *Only one instance of the game can exist in memory at any given time. This can be accomplished by creating unique identifiers for each instance of a game, team, or player.*

## [Design Constraints](#_1ksv4uv)

As a web based game, you will need a web designer/programmer, likely using HTML/CSS/JavaScript and a framework of choice as HTML/CSS/Javascript are the most populous languages on web browsers. The programmer working on this will have several responsibilities about how to manage the different web browsers that are popular amongst different devices.

You’ll also need a way to manage who hosts the game, the host will have their own individual port but once on the website you’ll want a way to navigate so all the players to that same port so they all receive the same information. This means that whatever different operating systems your client has will need to be able to use their based client-server models.

## [System Architecture View](#_44sinio)

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](#_2jxsxqh)

The UML for this game environment design featuring the core 4 object oriented principles primarily uses polymorphism and inheritance as it core pillars, while it does maintain some levels of encapsulation and abstraction; they are not the place where the heavy lifting of the code is handled; games, teams, and players all run through the entity object.

A brief overview of how each is applied is below.

Encapsulations : The Game Service, is private and static and all the other variables within the different entities (Game, Team, Player) are all private. Once a new team, game or player is created all of unique IDs are managed without user instruction and validated from private variables. For example

Abstraction : While no current demo users the program driver has access to the abstraction points are the getters, add methods, and strings which allow the user to only interact with adding a game, team or new player from the respective game. . In the code at this point that is all that is needed until we discuss a more user interface,

Inheritance : The current UML is structured of that every “GameObject” Instanced game. team or player maintains the same core structure of an ( Id, name) between multiple instances by using the entity class but will vary depend on each needs for example. Game has a list of teams, teams have a list of players and players have no lists.

Polymorphism : Finally polymorphism, because of the unique identifiers we can create multiple of the same objects thus running multiple games, with multiple teams and players, all that can be identified uniquely and checked before added to the list of available games.

**"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](#_z337ya)

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.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Server Side** | MacOS does offer a server side deployment option, but this is the highest cost purchase and either does not interact with other operating systems or does so poorly. | Linux/GNU based do off server based deployments and are open-source and free, and comes pre-installed with package managers. They can often be considered the fastest as well. The biggest downside is that they require someone who is specifically knowledgeable in it while the other two are more commonly ready. | WindowsOS deployment host, does have the advantage of being the most popular OS it will lack and will likely be most of the user base. But it does offer only a limited-free and premium model if it scales to the thousands of users and will likely incur costs. | There isn’t really a server side hosting option here not one with enough power to handle thousands of users. |
| **Client Side** | MacOS users will use safari at a much higher rate than any other user. Safari will behave differently to several common features used by the bigger windows and linux browses. This will take increased time to notice the difference between their browsers. | Both Linux and windows most popular browsers Firefox, Chrome, and IE for web browsing and similar, while there may be some smaller edits to be made on those other browsers, most of the code will be similar. | Both Linux and windows most popular browsers Firefox, Chrome, and IE for web browsing are similar, while there may be some smaller edits to be made on those other browsers most of the code will be similar. | The front end designer will need to be heavily versed to design for mobile web designs that will need to be included to all three platforms. There are a large variety of screens and web browsers for mobile devices more so than the standard computer platforms. Designing for these devices will be the most time consuming and biggest part of the client side development. |
| **Development Tools** | The development environment for all three major OS web browsers is similar. I’d develop using VScode with extension for any specific code language we needed.  Will also need Node.js or NPM for installing Javascript libraries. | The development environment for all three major OS web browsers is similar. I’d develop using VScode with extension for any specific code language we needed.  Will also need Node.js or NPM for installing Javascript libraries. | The development environment for all three major OS web browsers is similar. I’d develop using VScode with extension for any specific code language we needed.  Will also need Node.js or NPM for installing Javascript libraries. | Mobile development for web based environments is similar, but will require a specific set of content delivery as we are providing images to keep pages smaller and managed. |

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**: I recommend a Linux/GNU server deployment, the primary guideline for this will be cost. This will require a knowledge party on staff to handle and manage this deployment. It will reduce budget cost and functions on all the major browsers. All three server-side operating platforms are about equal in functional categories but the cost difference favors linux greatly.
2. **Operating Systems Architectures**: <Describe the details of the chosen operating platform architectures.> Linux runs a monolithic kernel architecture, it takes up more storage space in the operating system but runs faster than the windows which is the closest in comparable price. Since our storage requirements are very low its makes it an excellent choice.
3. **Storage Management**: Unless a player profile feature that tracks game history is implemented very little storage will be needed we can go as 64 GB (smallest SSD I could find )

Enough to store, out OS, template web pages, queue features, building new/join game actions and images for the games. We’ll also use an index based image reference system rendering images and clues to keep game build time as short as possible.

1. **Memory Management**: Server memory management will be based on how we manage two parts of the game that all players will need access to and which to cache and which to store, The Images and Game Variables (players, score, teams etc.). I believe it will be best for the game if the longest load times are in the initial launch, So the goal should be to access images from the server storage once per game then let the players hold the images on the browser. This will allow the image data to be in storage, and free up more Memory to store active games. We will also dedicate a section RAM of the handle requests if the server realizes it will not have enough memory to run the next game it’ll create a game queue until space free up. If the queues become too large and long, additional memory will be required.
2. **Distributed Systems and Networks**: I recommend using an apache http server as its the most popular web server to post, the individual users will make RESTful API requests to the database new games and updates to their individual games. It's open source and free and used pretty much everywhere. In a standard client server architecture model.
3. **Security**:

There isn’t any user private data to be collected, but we do want to keep each user's individual games private to who they invite to play with them. I think the easiest way to go about it would be giving the user a randomly generated code to join that would connect each user to the same game data for updates and lock access to updating that game behind the user using the room key. This won’t protect against users accidentally sharing their game rooms via stream but some of security must be placed on the user.