

Draw It or Lose It

**CS 230 Project Software Design Document**

Version 1.0

**Table of Contents**

**Document Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| 1.0 | 06/17/22 | Devin Hunter | This version added functionality such as making it so each game can have multiple, unique teams assigned to it and making it so each team is able to have multiple, unique players assigned to it. |

**Executive Summary**

Our new client, The Gaming Room, has hired our technology consulting firm to develop a version of their game called Draw It or Lose It. Draw It or Lose It is currently only available in an Android app, and our job is to expand the game to multiple platforms. Our solution is to develop a web-based version of the game so that Draw It or Lose It can be played via the web on a wide variety of devices.

**Design Constraints**

The design constraints for this project are technical constraints set by the client. The Gaming Room stated that each game can only have one instance in memory at any given time. We can use the singleton pattern in our code in order to abide by this constraint. Another constraint is that each team, game, and player must have a unique name. We can make sure this is the case by using the iterator design pattern to check whether a name has already been used within a game or team. Each team will be composed of multiple players and each game must have the ability to have one or more teams involved. In order to do this we must be able to check that every game, team, and player has a unique name and id and we must add the ability to create new teams and players for each game.

**Domain Model**

The Entity class is the base class from which the Game, Team, and Player classes inherit variables and methods from. The Entity class has id and name private variables which are important for it’s three subclasses to make use of due to the fact that each game, team, and player must have unique names and identifiers. The Entity class also has getter methods which it’s subclasses can use.

If we look at the second row of classes, we can see that the classes in this row all have relationships to one another. The GameService class can only have one instance in memory at a time and so it uses the singleton design pattern (as evidenced by it’s private constructor, it’s getInstance() method, and it’s private static instantiator). GameService also has other useful methods, such as addGame() to add a new game and getGameCount() to see how many games are going at one time. GameService can have 0 to many games with different names and Ids going at once, and each of those games can have 0 to many teams, and each of those teams can have 0 to many players. Each Game class has a method which allows new teams to be added to the game, and each Team class has a method which allows new players to be added to the team.

We can see that the ProgramDriver class contains our main() method, where we can create new games, teams, and players. The ProgramDriver class uses the Singleton Tester class’ testSingleton() method to make sure that there is only one instance of the GameService class in memory at any given time.



**Evaluation**

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** | A Mac would perhaps be the most expensive option on this list to use as a server. We would want an Apple device that can connect to ethernet to keep internet speeds fast, which likely means we would need an iMac. The main drawback to running a MacOS server is that we would have to use Apple’s proprietary hardware. The licensing for web hosting is also more expensive than Linux. | Linux is a popular option for web-based hosting. Linux servers are generally cheaper and more flexible vs. servers hosted on Mac or Windows because Linux is open-source. The hardware required in order to run a Linux server can also be cheap. Most Linux servers also do not need to go offline in order to apply updates or correct errors, which is nice. | A Windows server, like a Linux server, does not necessarily require expensive hardware to run on. Web-hosting service licensing can be expensive though. Windows server is also known to be more prone to hacking attempts and cyber threats. | Mobile devices can be used as servers, but are not as common a choice as Mac, Linux, or Windows. A mobile device would certainly take up less space. There is the concern that the device will lock, so the passcode would need to be widely known. It is also a concern that mobile devices connect via WiFi and an ethernet connection would be ideal for hosting. |
| **Client Side** | We can certainly develop the client side for Macs.  Developers would have to know Javascript, HTML, CSS and preferably a Javascript framework (such as Angular or Vue) in order to keep the code uniform. This is only if we want to make this a game that we can play in the browser.  If we wanted to develop an app for macOS, then we’d need to develop the app using Swift and Xcode, and we’d need to do this on Apple hardware, which is generally expensive. | App development for Linux requires a different kind of expertise than software development for a Mac or Windows machine because developers need to be familiar with the Linux terminal.  If we want to develop an app for Linux systems then hiring developers for the project may be more expensive because they’d likely need to be at least a little experienced in Linux.  Otherwise we can use Javascript to create a browser-based game, which requires less hands-on experience with Linux. | If we’re developing an app for Windows machines, out developers would need to be proficient in C# or C++.  If we’re developing a browser-based game developers would have to have experience in Javascript, HTML, CSS, and Javascript frameworks. | There are lots of options for different mobile devices. For a browser-based application developers would have to be familiar with the mobile-first approach to development and certainly make their code responsive so that it can fit any mobile device screen width. We can use Javascript for the client side and easily make the application responsive so that it fits all screen sizes.  Otherwise, if we wanted an app for Apple devices in addition to the Android app The Gaming Room already has, we’d need developers who are familiar with Swift. |
| **Development Tools** | Xcode is a popular IDE for Mac. It can be installed via the terminal or on Apple’s website. You can also download VS Code for Mac although it is designed by Microsoft.  We can design the server using Python in the Xcode IDE. We could also install Node.js on our Mac and use the VS Code IDE. We could also create the server using Java and download the Eclipse IDE for Mac. We could also develop the server in Ruby using RubyMine for Mac.  For client-side development I would recommend coding in Javascript using a framework such as Angular or Vue to keep the code uniform and using VS Code as the IDE if we want to be able to play the game in a browser. If we want an app for macOS then I recommend using Xcode and Swift. | Python comes pre-installed on Linux, so it could be a good choice to use for developing the server side or the client side of Draw-It or Lose-It. We could also develop the server side in Java on Linux using the IntelliJ IDEA IDE or Eclipse.  Visual Studio Code is also available for Linux so we could develop the client side in that IDE using Javascript and a JS framework if we wanted to be able to play the game in a browser window. | For the server, we could potentially use Java, Javascript (node.js), Ruby, Python, or C#. For Java, we could use the Eclipse IDE. For Python we could use Visual Studio or PyCharm. For Javascript I would recommend VS Code. I could also recommend Visual Studio or VS Code for C#.  I personally use Visual Studio for C++ if that’s the language we want to use. For Ruby, RubyMine could be a good choice.  For client-side development I would recommend coding in Javascript using a framework such as Angular or Vue to keep the code uniform and using VS Code as the IDE if we want users to be able to play the game in a browser.  If we want a Windows application then I would recommend developing the client side in C# or C++. | The Gaming Room already has an Android app.  If we wanted an app for ios we’d need to develop the app using swift and Xcode on a Mac.  If we wanted a mobile browser-based game we’d need to develop it using Javascript and a framework like React, Angular, or Vue, and we’d need to make sure that the game is responsive to all screen sizes and shapes. |

**Recommendations**

Analyze the characteristics of and techniques specific to various systems architectures and make a recommendation to The Gaming Room. Specifically, address the following:

* **Operating Platform**:

I would recommend an OS in the Linux family for the Draw It or Lose It server. The Gaming Room is a business, so naturally they will want to cut down on costs. Hardware prices for a Linux server are flexible and The Gaming Room can keep hardware costs as low as their budget requires (while also taking into account the fact that due to the hardware’s modular nature, they can add RAM and storage as needed, unlike if the server was hosted on a Mac). The cheap cost of a Linux server also leaves more room in the budget for the Gaming Room to be able to have multiple servers if needed. Linux servers often do not even need to go offline to apply updates or correct errors, which means less server down time, and less server down time means more money saved.

* **Operating Systems Architectures**:

Linux is a free, open source OS. Due to it’s open-source nature, Linux has a large community behind it, but some distributions also lack the quality of enterprise support that Windows and macOS have. Some Linux server distributions such as Ubuntu server and CentOS are free, and some have a price tag such as the Red Hat and SUSE enterprise server distributions. The hardware needed to run Linux can be as cheap or as expensive as needed. Due to the popularity of Draw It or Lose It, The Gaming Room may want to add hardware capability over time, and due to the fact that Linux OS does not require any rigid proprietary hardware to run on, that is entirely doable. Linux users need to be familiar with Linux commands in order to be able to make full use of the OS.

* **Storage Management**:

The Gaming Room can have as much or as little storage as they require due to the modular nature of the hardware a Linux server can run on. I would recommend using SSDs for storage rather than HDDs due to the fact that HDDs have moving parts and are more prone to failure/wear and tear over time. SSDs are generally faster and the storage capacity of SSDs these days is sufficient to store the amount of information The Gaming Room needs to store (including user identification data, user statistics, all of the image files for the game, etc.).

* **Memory Management**:

The Draw It or Lose It software needs to be able to access images quickly in order to render the images for players at an engaging pace. If this process is slow, the game could quickly become uninteresting to players due to the amount of waiting involved and player engagement on all platforms would likely decline. In order for the game to remain fast and fun, the game’s image files must be available in RAM so that they can render almost instantaneously. Draw It or Lose It could employ dynamic loading so that any images that the game recognizes it will not be using in that specific round are not moved from storage to memory. This could make it so that load times are faster and the game’s desired pace is maintained.

As with storage, The Gaming Room can increase or decrease the amount of RAM they’d need to fit their needs due to the flexibility of the hardware a linux server can run on.

* **Distributed Systems and Networks**:

Linux servers often do not need to stop in order to do maintenance or apply updates, this means little downtime (which is important for maintaining player engagement). The hardware for the server would need to have an ethernet cable connection rather than WiFi in order to maintain the most reliable connection. During the game, data and files (such as image files) could be requested from the server by the client in the form of GET requests and data could be updated with POST and PATCH requests. Due to the fact that the game is web based, games running on all platforms can access the server (provided the player has an adequate internet connection on their end).

* **Security**:

User authentication (with username and password upon login) could keep user data secure. I would not recommend using multi-factor authentication upon login due to the fact that users need to stay engaged and that would likely cross the line to where logging in becomes inconvenient enough to turn some users off to the game. User permissions should be split up by role (such as administrator, guest, general user) so that basic users cannot access critical information or change things that they should not have the power to change.