

Draw It or Lose It

**CS 230 Project Software Design Template**

Version 1.0

**Table of Contents**

**Document Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| 1.0 | 05/19/2023 | Alexander Dollison | Initial version, summary and requirements obtained, first prototype of game created. |

**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**

Our Client, The Gaming Room, wants to develop a web-based game that is available on multiple platforms, called Draw it or Lose it. Currently, it is only available on the Android platform, and the game consists of several teams that must guess what images are being produced. The program must support more than one team, with each team having several players. Game names and team names must be unique, which each instance being unique, as only one instance can exist in memory at a time.

**Requirements**

*-As listed above, the game must support multiple teams and multiple people on each team.*

*-Must run on multiple platforms.*

*-Teams and game names must be unique.*

*-Only one instance can exist in memory at a time.*

**Design Constraints**

- Each game must have the capacity to store more than one team.

- From each team, there must be capacity for more than one member.

-The game name andteam names must be unique to allow the system to check if that name is in use.

- Only one instance of the game can exist at a time.

-Game must be capable of running on multiple platforms.

**System Architecture View**

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**

The UML diagram below provides a visual design of how the gaming system will be developed. The Entity class is a base class, and holds common attributes and behaviors. It is also a parent class for the other classes named Game, Team, and Player. This exemplifies inheritance, explained by an open arrow. The 3 classes, along with GameService class all share the same multiplicity, which explains that these classes may contain zero or many objects. The GameService class hols most of the original methods that make up the game, which allows us to add multiple players to each team and multiple teams to each game. These are identified by a unique name and id. We can see two separate classes, ProgramDriver and SingltonTester, which are methods that work together to test the code. ProgramDriver calls upon SingletonTester to test the code, ensuring only one occurance of the game is running at a certain time. This fulfills the requirement of the program to test if one game instance exists in memory. This is also called Abstraction, which tests functionality of the program. Polymorphism is the understanding that all teams, players, and game names are the same considering the attributes available to them. The objects themselves are unique only because of their personal identifiers, which create objects that they are assigned to.



**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** | Mac is stable with hosting of servers, but can take more time and special instructions to set up. It comes with apple products by default, and only apple products can run this OS. | Linux is the basis for many open-source OSs, and is thus the most compatible with the widest array of products. Because of this, development for server-side software is always being worked on. | Windows is the most versatile and common OS, and is the default for most server-side programs. As such, it is the easiest to set up for servers, but it can also be constraining, as the performance of Windows-based games and systems are lacking compared to other OSs. | Mobile devices cover an array of different OSs that have been simplified, so making a game that is compatible with mobile devices in general means that you have to consider multiple OSs as is. You must also consider performance, since mobile devices are typically the least-powerful devices out there. |
| **Client Side** | Mac is a very specialized OS, and as such it can take longer and cost more to develop products for Mac, especially when considering the cost of Mac products themselves. | Linux is more widely available, cheaper to make products for, but is not as popular as the other three types of OSs, and as such may not make as much money, nor see as much support. | Windows is widely used, so expertise for making a game that runs on Windows is found practically everywhere. They can be cheap to produce and likely won't take as long as development for other OSs. | Mobile devices are all constrained by their size and power, so making a game that runs on them doesn't cost as much as for the other OSs, but can be far more limiting. Because their are different OSs for different mobile devices, you'll need a team of experienced software engineers to keep up with the demand of a game made for this OS. |
| **Development Tools** | Macs can use python IDEs, Eclipse, Visual Studio, PyCharm, Android, and many of the other tools used by other OSs because of the support given to Mac producst in recent years. | There are a few IDEs like Eclipse and Visual Studio that also support Linus, however, they lack much of the overlap that Mac and Windows users have, and is thus more limited. | Windows contains many of the IDEs the others have, as it is the most used and tends to be the basis for many of the tools like Visual Studio and Eclipse, as well as IDLE, and Android Studio. | Many of the IDEs used for developing mobile devices are found only on personal computers or laptops, Windows, Linus, and Mac devices. As such, developing on a mobile device, while not impossible is impractical. |

**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**: If we expand to Microsoft Windows, which has tools that allow easy translation to MacOS, Linux, Android, and other OS types, it will be easier to design a program which works on all of said platforms with minimal issues.
* **Operating Systems Architectures**: Microsoft Windows has the largest number of IDEs that translate well over to other OSs, and as such is the most compatible OS for multi-platform gaming.
* **Storage Management**: Windows provides access to cloud-based data sending and receiving, as does Mac and Linux, and will allow teams to be formed over the internet by connecting to a host, rather than relying on a program to provide every connection by itself.
* **Memory Management**: Because Windows can use cloud saving, sending, and receiving, it can easily overwrite information when needed and send out new information quickly. It also allows information to be accessed by other users in a neat and quick package that allows high performance.
* **Distributed Systems and Networks**: By having a host that uses an IDE such as Eclipse that is compatible with most if not all of the platforms, disconnects and errors between platforms can be resolved before they reach the user and before the program launches. Drops in connectivity would be on the user's end, if for instance a user lost internet, but if done in the middle of a game, we wouldn't want other players to lose connection because of this, so having a player as a host rather than a separate, temporary host wouldn't be a good option.
* **Security**: Security is potentially the most important aspect in a game or in software, as cheaters and hackers can potentially obtain IPs and information of users through the software if proper steps are not taken. Using anti-cheat systems that detect unusual activity of users or software that can catch the use of other software that is not approved by the company for the game would be the most reliable method for protecting users. Protecting the company, however will involve keeping backups of the system and specific programs that can detect interference with the system and the host running the software. Having temporary hosts can mean that players that are accessing information they shouldn't have can be kicked or that the game can be shut down quickly without risking information sent or received from other temporary hosts.