Draw It or Lose It

CS 230 Project Software Design Template

Version 1.0

Table of Contents

Version	Date	Author	Comments
1.0	01/25/25	Malcolm Marberry	Completed the executive summary, requirements, design constraints, and domain model sections.
1.1	02/09/25	Malcolm Marberry	Completed the evaluation section and added References
1.2	2/21/25	Malcolm Marberry	Added and completed the recommendations section

The Gaming Room wants to create a game called Draw It or Lose it which is inspired by the 1980s television game *Win, Lose or Draw* where teams compete to guess what is being drawn. Unlike the show, images are rendered from a library of stock drawings at a steady rate—finishing after 30 seconds. After failing to guess correctly, other teams can each submit their own guesses within 15 seconds. Any number of teams can participate, and multiple players can be assigned to the same team; the only restriction being that team names must be unique.

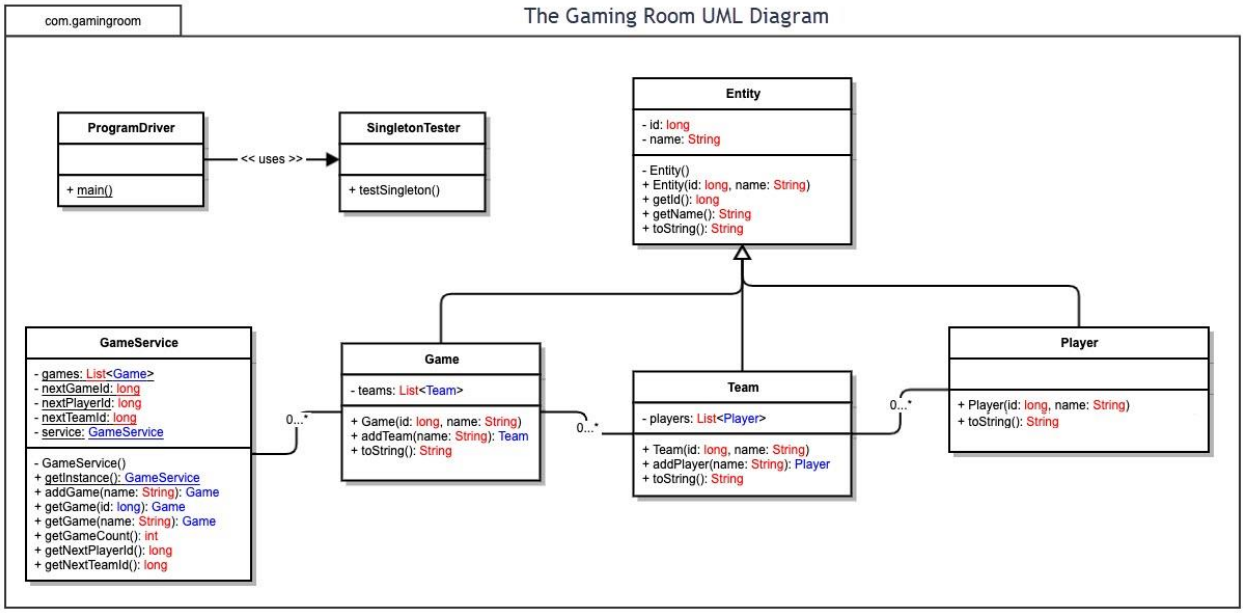
Requirements

- Games can have multiple teams
- Teams can have multiple players
- Game and team names must be unique so that players can check to see if a team name is taken
- Only one instance of the game can exist in memory at any given time

Design Constraints

Since the game is web-based, there are browser-related design constraints. While it's not specifically mentioned in the client's requirements, it'd be important to consider potential compatibility issues. Multithreading might be necessary to ensure that many teams with any number of any number of players can all use the program at the same time with as little latency as possible. Another concern is how exactly the database of stock images would be referenced by the program—if, say, the client planned on adding photos over time, the app would need to account for a growing database.

The diagram shows how the "objects" of the program are designed in relation to each other. Starting at the top, the Entity class which is inherited by Game, Team, and Player shows the OOP (object-oriented programming) principle of inheritance by defining the common members of the three child classes—for example, they all have an "id" and a "name". Moving down, a key part of the diagram is the relationships between the bottom four classes. The "0. . .*" between them describes a "zero to many" relationship which means that a Game can have any number of Team objects and a Team can have any number of Player objects. This matches the client's requirements for the program by ensuring that there can be multiple teams with multiple players. Another key detail is that the GameService class is a Singleton, meaning there can only be one GameService object at a time, which matches the requirement of only one instance of the game existing in memory at any given time.


com.gamingroom
The Gaming Room UML Diagram
ProgramDriver
+ main()
<< uses >>
SingletonTester
+ testSingleton()
Entity
- id: long
- name: String
- Entity()
+ Entity(id: long, name: String)
+ getId(): long
+ getName(): String
+ toString(): String
GameService
- games: List<Game>
- nextGameId: long
- nextPlayerId: long
- nextTeamId: long
- service: GameService
- GameService()
+ getInstance(): GameService
+ addGame(name: String): Game
+ getGame(id: long): Game
+ getGame(name: String): Game
+ getGameCount(): int
+ getNextPlayerId(): long
+ getNextTeamId(): long
0...*
Game
- teams: List<Team>
+ Game(id: long, name: String)
+ addTeam(name: String): Team
+ toString(): String
0...*
Team
- players: List<Player>
+ Team(id: long, name: String)
+ addPlayer(name: String): Player
+ toString(): String
0...*
Player
+ Player(id: long, name: String)
+ toString(): String

Evaluation

Development Requirements	Mac	Linux	Windows	Mobile Devices
Server Side	Characteristics: Unix-based system (Darwin) (Silberschatz et al., 2008 ch. 2.7.3) and built-in support for Apache, PHP, and programming languages like Python Advantages: compatibility with IOS, Homebrew package manager for importing community resources (Homebrew), high security because of its Unix-based system, and "traces system calls" using ktrace (Silberschatz et al., 2008 ch. 2.12) Weaknesses: "Mac servers tend to be more expensive than similar Windows Server configurations" and limited software compatibility (Whatley, 2024).	Characteristics: written mostly in C (Silberschatz et al., 2008 ch. 2.6.3), highly customizable, VMware workstation, and open source Advantages: VMware workstation lets you run "guest operating systems" for testing cross-platform compatibility (Silberschatz et al., 2008 ch. 2.8.6.1), supports a wide range of programming languages, it's open-source and free which reduces costs, and according to Tsurkan (2024), the "ideal choice for. . .cloud services" (Tsurkan, 2024) like AWS, Azure, and GCP. Weaknesses: potential compatibility	Characteristics: supports multiple languages and frameworks, integrated into all Microsoft-related technologies, and has VMware workstation Advantages: testing other operating systems on virtual machines, directly compatible with SQL databases, and "the interface. . .is built into the kernal and system libraries" (Silberschatz et al., 2008 ch. 2.6.2) meaning access to GUI tools for server management. Weaknesses: costs more, on average, than a Linux-based server (adding up server licenses and other software like SQL servers), generally slower and more resource-heavy than other options, and its position as the most used	Characteristics: Not a common option for hosting servers, so there aren't many deployment methods. Mobile operating systems, like IOS for iPhones, are usually specialized for client-side applications. Advantages: portability, less costly than other options, and their portability and ease of use make them valuable for testing small-scale deployment. Weaknesses: not designed for major applications, OS restrictions—IOS, for example, doesn't support as many languages, and wouldn't have the processing power of devices that can run other operating systems

		issues with Windows clients and a higher knowledge barrier for development than other operating systems	operating system makes it a target for security attacks	
Client Side	Other than using standard practices, Apple claims the best way to develop a web app for Mac OS is through the Safari browser (Apple, 2024). Any webpage can be converted to a web app, so if the front-end works in Safari, there shouldn't be any additional costs, time, or expertise needed.	Different distributions of Linux may have different compatibility requirements. For example, Red Hat Enterprise Linux, which has its own "Application Compatibility Guide" (Red Hat Enterprise Linux 8, 2024). It'd be important to test several of the most popular distributions for compatibility.	Microsoft provides a list of requirements for deploying clients to Windows computers (BalaDelli et al., 2024). This includes requirements for the app to work as intended as well as security features needed to comply with the "Windows firewall". Developers of the client-side would need to reference this document.	Mobile devices more often vary in screen size—something for development to account for. There could also be performance issues/differences; certain processes may be more time or power consuming on a mobile device. Also, there's generally less available memory and storage on mobile devices, so these processes would need to be resource efficient as well.
Development Tools	The technical requirements of Mac OS web applications aren't that different from those of Linux or Windows. It supports many of the most popular programming languages for server and client-side, and it comes with many free IDEs and other development tools. That said, the "Apple Developer Program" which	As previously mentioned, there are multiple distributions of Linux—all with their own compatibility requirements. This could strain the development team as they attempted to account for as many distributions as possible. There may need to be more teams depending on how many different	There may need to be a separate team for developing the Windows client depending on how strict Microsoft's requirements are. It has access to many free IDEs and development tools, but in order for apps to be in the Microsoft store, you need a subscription. Tools like Visual Studio,	The two most popular mobile operating systems, Android and IOS, both have their own dedicated IDEs for development. The former has Android Studio provided by Google, and IOS has Xcode provided by Apple. Both of which are free, so there wouldn't be any additional licensing costs. Development team(s) would likely need proficiency in

	lets the app be on their App Store is 99 USD a year (Apple).	compatibility constraints arise because of this. Linux has access to the same free IDEs as other operating systems, and it has several app stores. Most of the former require a license, which adds to the cost of development.	Bootstrap, or React are available for free as well as every popular programming language.	each operating systems official IDE to ensure compatibility. Depending on how many IDEs this includes, multiple teams may be needed.

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 using a Linux-based server.

2. **Operating Systems Architectures**: Linux is one of the industry standards: being open source means it's highly customizable, it's generally cheaper than other options, and it's the preferred operating system for many cloud services like AWS, Azure, and GCP. The next best option, which in my opinion is Windows, would likely be more costly, slower, and need more resources (memory), which could conflict with the app's present memory constraints relating to image rendering.

3. **Storage Management**: The app's current image collection is 200 8-megabyte files, in total requiring at least 1.6 terabytes of storage. To safely store them, along with other relevant resources, I'd recommend a minimum of 2 terabytes of general-purpose storage (ufs or zfs), although, as will be discussed in memory management, more could be beneficial. A supplemental JSON file should be used to track image files because of its portability and compatibility with client-side programming languages like HTML and JavaScript.

4. **Memory Management**: Image transfer and render speeds are the main memory constraints of Draw It or Lose It. Linux's implementation of "swap space" allows for more deliberate memory management (Silberschatz et al. 2008, ch 12.6.2). Silberschatz et al. state that, in general, swap space is used to free up memory by moving the less demanding processes (among other things) to disk space. Linux allows the use of "multiple swap spaces" which spreads the burden and the choice between "raw partitions" and "file-system space" for swap space. These features give more options for memory management and would increase performance.

5. **Distributed Systems and Networks**: Each game of Draw It or Lose It will need to connect to several clients, and potentially several platforms, at once. This can be accomplished by hosting the "Game" --teams, players, scores, current images, etc. --on the server and sharing the relevant information to clients via the network. This presents concerns about varying internet speeds, outages, and other potential complications. There would need to be a series of "checks" ensuring that individual players' internet speeds didn't affect the game: waiting for everyone to load before starting a game, preloading images to memory, and removing a player from a game if they lose internet connection.

6. **Security**: User accounts for Draw It or Lose It contain sensitive information like first and last names and email, so server's user database needs to be secure. The server needs to ensure that clients only receive information they're authorized to have, and there needs to be safeguards against security attacks. For example, Silberschatz et al. (2008) describes a "buffer-overflow attack" which can "run between systems and can travel over allowed communication channels" (Silberschatz et al. 2008, ch 15.2.4). Linux supports the "NX feature" of recent CPUs which was designed to combat this type of attack.

References

Apple. (2024, November 18). *Use Safari Web Apps on mac*. https://support.apple.com/en-us/104996

Apple. (n.d.). *Enrollment*. Enrollment - Support - Apple Developer. https://developer.apple.com/support/enrollment/#:~:text=The%20Apple%20Developer%20Program%20annual,currency%20during%20the%20enrollment%20process.

BalaDelli, alexbuckgit, mestew, aczechowski, & dougeby. (2024, March 28). *Windows client prerequisites - configuration manager*. Windows client prerequisites - Configuration Manager | Microsoft Learn. https://learn.microsoft.com/en-us/mem/configmgr/core/clients/deploy/prerequisites-for-deploying-clients-to-windows-computers

Homebrew. (n.d.). https://brew.sh/

Red Hat Enterprise Linux 8. (2024, March 25). *Red Hat Enterprise Linux 8: Application compatibility guide*. Red Hat Customer Portal. https://access.redhat.com/articles/rhel8-abi-compatibility

Silberschatz, A., Galvin, P. B., & Gagne, G. (2008). *Operating System Concepts* (8th ed.). John Wiley & Sons, Inc. February 5, 2025, https://learning.oreilly.com/library/view/operating-system-concepts/9780470128725/silb_9780470128725_oeb_c14_r1.html#h1

Tsurkan, A. (2024, January). *Why clouds use linux*. Codefinity. https://codefinity.com/blog/Why-Clouds-Use-Linux

Whatley, S. (2024, March 27). *Comparing mac and windows servers for web hosting*. MamboServer. https://www.mamboserver.com/blog/comparing-mac-and-windows-servers-for-web-hosting/#:~:text=Mac%20servers%20tend%20to%20be,to%20their%20stability%20and%20reliability.