

Draw it or Lose it

# **CS 230 Project Software Design Template**

Version 2.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc88920038)

[Table of Contents 2](#_Toc88920039)

[Document Revision History 2](#_Toc88920040)

[Executive Summary 3](#_Toc88920041)

[Design Constraints 3](#_Toc88920042)

[Domain Model 3](#_Toc88920043)

Evaluation 4

## Document Revision History

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 11/11/2021 | Kareem Elkwae | Completed executive summary, design constrains, and domain model. |
| 2.0 | 11/27/2021 | Kareem Elkwae | Completed the evaluation section. |
| 3.0 | 12/12/2021 | Kareem Elkwae | Completed recommendation section. |

## Executive Summary

We will design a program that will render an image over 30 seconds. The player/user will have 60 seconds starting from the beginning of the image rendering to guess what the image is. If the player does not guess the image in the 60 seconds, the other team will have 15 seconds to guess what the image is. Every game will be able to have multiple teams, each team will be able to have multiple players, game and team named must be unique so that teams will be able to check if that name is available, and only one instance of the game can exist in memory at any time.

## Design Constraints

Issues that may arise during the development of the game are making sure that each team and game name are unique, counting the time of play, rendering the image at a steady rate, and ensuring one instance of each game, team, or player is in memory at any time.

## Domain Model

The ProgramDriver Class holds the main code which is currently used for testing. The SingletonTester class is also being used to test that the other classes’ singleton pattern has been set up correctly. The Program driver uses the SingletonTester for its testing. The GameService class is responsible for running the games. Only one instance of it is allowed in memory at any time. The GameService class holds a list of games in its instance. Each game has a unique identifier and name. It also contains functions to search for or add a game to its games list by iterating through the list. The Game, Team, and Player classes are all extensions of the Entity class. Each of the three inherits attributes and common functions. Each of the three, contains their own constructor and their own print control that overloads the entity “toString” function. The Game class also holds a list of teams, and the Teams class holds a list of Players. The same iteration pattern happens when adding players or teams to their respective lists to ensure only one instance of a team or player in the list exists with the chosen name. For each Game, there can be 0 or more instances of a team, for each instance of team, there can be 0 or more instances of player.

**"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

Based on my experience and knowledge about the various operating platforms, I will evaluate the strengths and weaknesses of each of the common platforms in the table below.

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side** | Mac has great compatibility and support for other Mac devices. The Mac terminal also makes server-side configuration changes easy. One downside is that Mac systems are generally the most expensive. | Linux is an inexpensive platform. It is commonly used due to its ability to be compatible as well as the fact that it allows for easy server-side changes. | Windows is very commonly used. It allows for easy compatibility with windows machines and has a little less of a learning curve. There is also a greater amount of knowledge and support available. | Mobile devices are probably not the best choice for web hosting due to the limited hardware availability and the complex nature of implementing web hosting on a mobile device. |
| **Client Side** | As mentioned above, Mac systems tend to be more expensive. Macs tend to require more technical expertise than windows but less than Linux. It can sometimes be moderately time consuming to implement. | The Linux platform is relatively inexpensive, especially when compared to Windows and Mac, but more technical expertise is required to support Linux client side. | Windows systems can range from inexpensive to very expensive depending on the device specifications. It can be pretty easy to support due to the common use of the platform, and generally would not be too time consuming to implement. | The various mobile device platforms allow for more accessibility to client-side systems. They can range in cost from very low to very expensive. Mobile platforms can also be technically difficult to implement on. |
| **Development Tools** | xCode, Visual Studio, Eclipse, PyCharm, Java, C++, Python, Swift, SQL, HTML, C#, PHP, C, Ruby, JavaScript, CSS, and Objective C are some of the more common tools used to deploy on Mac. Many of these tools are inexpensive or free. There may be a need for different teams with different areas of expertise to work together. | Visual Studio, Eclipse, PyCharm, Python, C++, Java, HTML, CSS, SQL, PHP, C, C#, Swift, Ruby, JavaScript, and Objective C are some of the available tools used on deploy on Linux. All of these tools are relatively low cost or free. There may be a need for multiple development teams with different areas of expertise to work together. | Visual Studio, Eclipse, PyCharm, Python, C++, Java, HTML, CSS, SQL, PHP, C, C#, Swift, Ruby, JavaScript, and Objective C are some of the available tools used to deploy on Windows. All of these tools are relatively low cost or free. There may be a need for multiple development teams with different areas of expertise to work together. | Visual Studio, Eclipse, PyCharm, C++, Java, Python HTML, CSS, PHP, C, C#, Swift, and JavaScript are some of the available tools used to deploy on Mobile devices. All of these tools are relatively low cost or free. There may be a need for multiple development teams with different areas of expertise to work together. |

## Recommendations

1. **Operating Platform**: I recommend using linux for the server.
2. **Operating Systems Architectures**: I would recommend linux due to the stability, security, and compatibility that linux offers. There is a requirement for more technical knowledge to implement on it, but it should be doable for a qualified developer.
3. **Storage Management**: We need to ensure that the storage is formatted correctly. It will need to store the 200 pictures, the code, and game records, and it will need to be easily and quickly accessible. I recommend a direct access system with index pointers so that when files are needed, they can be quickly and easily called.
4. **Memory Management**: It is important to have a good amount of memory in the system. This is because of the fact that we will want to be able to support many instances of the game. We will also want to be able to have game resources cached, so that they can be quickly loaded when needed for the gameplay. We also want to make sure that unneeded resourced are clear from memory to open those resources up.
5. **Distributed Systems and Networks**: To offer support for Windows, Mac, and Linux, the end user side of the game should be available via web browser. This is because there are server web browsers that operate on all three operating systems so compatibility can be ensured. The web client will interface with the back-end server. If we would like to offer support for mobile applications, we can create an application for iOS and Android so that mobile devices can interface with the game as well. They would behave the same as the browser model and the server would do most of the work. If there are any outages system issues, we can have a backup system so that players can continue to play if the hardware goes down, otherwise, if there is an internet outage, we can implement a backup internet line to ensure that we can maintain connectivity.
6. **Security**: I recommend we use best practices for security. Firstly, the linux server architecture is generally viewed as more secure than others. we will implement encryption for any logins. We will not request any data from players other than their login, and we will ensure that our servers are protected by segregating important information.