

Draw It or Lose It

# **CS 230 Project Software Design Template**

Version 1.0.1b

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc115077317)

[**Table of Contents 2**](#_Toc115077318)

[**Document Revision History 2**](#_Toc115077319)

[**Executive Summary 3**](#_Toc115077320)

[**Requirements 3**](#_Toc115077321)

[**Design Constraints 3**](#_Toc115077322)

[**System Architecture View 3**](#_Toc115077323)

[**Domain Model 4**](#_Toc115077324)

[**Evaluation 6**](#_Toc115077325)

**Recommendations 9**

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 01/19/2023 | Rachel Aldava | Preliminary foundations for the game |
| 1.0.1a | 2/5/2023 | Rachel Aldava | Slightly revised the evaluation section. |
| 1.0.1b | 2/19/2023 | Rachel Aldava | Revised recommendations |

## [Executive Summary](#_sbfa50wo7nsh)

Creative Technology Solutions is requesting a game in which players, competing in teams, will try to guess what an image is while it is in the process of being rendered. Because this game will be multi-player over the internet, one notable challenge will be to facilitate communication between the devices. This will be a multi-platform game which seeks to be cross-compatible.

## 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](#_2et92p0)

CTS will need to procure images along with the rights to use those images.

The program will need to interface with servers which will house the library of images. While it may be possible to have a stock-image vendor provide the hosting service for the images, it is unlikely that they would be capable of doing so with sufficient reliability. Therefore, a server which hosts the library will need to either be built or rented.

Additionally, because this is a multiplayer game, there will also need to be another server which will handle match-making and can initialize a game. While It may be possible for the individual games to be ran solely on the players’ devices, developing a decentralized game environment would require a substantially increased development and testing budget due to problems which may arise when in one hypothetical game, a Mac tries to communicate with two Androids, an iPhone, a Windows, and a heavily modified smart toaster running an esoteric version of Linux. It may be simpler for the match-making server to also handle running individual game environments.

Setting up both of these servers will take a substantially different skill set from our coding team, and while it may be possible to use the same language for all devices involved, there may be performance benefits to using languages which are more suitable to the device’s role in the game. If multiple languages are being utilized, we may need to split our programmers into different development teams.

Depending on the expected number of users and the number of images which are planned, it may be possible to have one server handle both match making and image hosting.

While some code could be recycled for different operating environments, separate development teams will need to be created for each platform which we wish to support.

## [System Architecture View](#_ilbxbyevv6b6)

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](#_8h2ehzxfam4o)

The ProgramDriver will interact with a piece of code called SingletonTester, which currently runs a series of tests in order to ensure that the other parts of the code are running correctly.

At the top of the diagram, an Entity defines some basic components that Game, Team, and Player will have in common. For example, a Game would be a kind of Entity, and since it is an entity it will have an id number and a name, as well as a description whenever toString() is used by other areas of the code. An important aspect to keep in mind is that this code will allow the creation of an entity which is neither a Game, Team, or Player; this may be an oversight in the code. A potential solution may be to change Entity(id, name) from public to protected. This would mean that an Entity object could not be created except within the code sections which contain Game, Team, and Player.

There can be zero to many Players to a Team and a Player is created within the Team code section. Likewise, there can be zero to many Teams to a Game and a Team is created within the Game code section. Finally, there can be zero to many games occurring at any point in time and a Game is created by the GameService section of the code.

The GameService section of the code will allow the creation of one and only one GameService object which does NOT inherit any features that an Entity might have. This GameService object handles the creation of Games and houses information about the ID numbers which are available to be assigned to a Game, Team, or Player.

GameService, Game, and Team have lists of current Games, Teams, and Players respectively. When a higher order object creates a lower order object, it will check if the requested parameters are valid. In this way, the code will check for name duplication.

**"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](#_2o15spng8stw)

| **Development Requirements** | **Mac** | **Linux** | **Windows** | **Mobile Devices** |
| --- | --- | --- | --- | --- |
| **Server Side**  The server(s) will need to have the capability of connecting to a large number of users, properly supporting and running the game environment including all of the game instances, and hosting a large library of data.  This system will need to be scalable and have a high amount of uptime. In order to minimize downtime, this server must conform to industry standards rather than novel solutions. | I am not aware of any current server operating system which is offered by Apple. Any server using MacOS would need to be a retrofitted personal machine, and thus poorly optimized for the task; such a system would not align with industry standards. | Using Linux for a server is a very common choice and virtually all system admins will have experience using it. While usually free, some versions, such as Red Hat, require a license but they also offer customer support.  Choosing between a free or customer-support model is a business decision. Choosing a free version exposes the company to an increased risk of longer downtimes and could potentially be counterbalanced by employing a higher skilled, and more expensive, systems administration team.  In either case, this architecture should meet the requirements. | A Windows server seems to be a less common choice which I have no direct experience with. While there is a licensing fee (exact pricing is not readily available), they do offer customer support for their products. Many industries successfully implement Windows servers.  It would appear this architecture could meet the requirements, given that the licensing fee is acceptable. | Using an old Android phone with a custom operating system as a server IS possible, but due to limited hardware resources, battery dependency, and wireless connectivity, it is absolutely unsuitable for any hosting task; such a system would not align with industry standards. |
| **Client Side**  Since non-mobile platforms will use a web browser, the code could be designed in such a way as to not rely on OS-specific resources. If this is the case, development time, cost, and expertise would be shared across Windows, MacOS, and Linux, however, testing expenses would increase with each supported platform; I would highly recommend avoiding unnecessary code which would be OS-specific.  The top 5 browsers should be tested on all supported on non-mobile devices. Special attention should be paid to Chrome, which holds an overwhelming 77% market share.  Minor web browsers such as Netscape may be a poor investment to support, given the limited userbase | If MacOS resources are called, the MacOS platform seems to share a language with IOS, so there may be degree of cross-compatibility over certain portions of the code, however since there is a dramatic difference between the architectures from the CPU to the OS, there will still be a large portion of the code which will need to be unique to this OS. While development for MacOS is not a common as Windows, it is still a common skill for many developers. | If Linux resources are called, as with MacOS there may be some degree of similarity between Andoid and linux structures, however if the app was developed in an android specific language these similarities may be more difficult to leverage.  Largely, I would expect any original code developed for the Linux platform to be vastly more expensive than other platforms, as there are many flavors of Linux and there are not as many developers with experience creating client-side applications. | If Windows-specific resources are called, there exists a large pool of developers who are familiar with developing for the Windows platform.  While it is likely there will be no code to recycle, I would expect this platform to be among the cheapest and most profitable to maintain given its large market share and developers’ existing familiarity with the architecture. | Given that the android app has already been developed, much of the original specifications, designs, and user-interface elements may be recycled. However, the underlying code will need to be re-written as IOS and android do not share a common language. Porting an android app to the IOS platform is a common task and does not require any unusual skillsets.  While it may be somewhat costly, I would try to prioritize supporting this platform given its similarity to Android in terms of UI and target audience and given IOS’s market share. |
| **Development Tools**  The Eclipse IDE purportedly supports HTML development, although I have no first-hand experience developing HTML with Eclipse.  There will need to be one development team for supporting the HTML-based web browser platform which will include support for the different browsers and (potentially) coordinate with sub-teams of OS-specific client support.  There will need to be another development team to handle server-side deployment.  The Eclipse IDE appears to be free, however they do seem to strongly offer membership packages for organizations.  X code appears to be free; I cannot find any mention of commercial licensing.  Visual studio is free for commercial use per their FAQ section.  Jetbrains seems to be mostly free for commercial use, and in the context of this particular project it would seem to be free. | MacOS will run Java, Swift, Objective C, and many other programing languages.  Possible Mac compatible IDEs could be Eclipse for Java and Xcode for Swift and Objective C. | Linux will run Java, C++, Python, and many other programming languages.  Possible Linux compatible IDEs could be Eclipse for Java and C++ and PyCharm for Python. | Windows will run Java, C++, C#, and many other programming languages.  Possible Windows compatible IDEs could be Eclipse for Java and Visual Studio for C++ and C#. | Android can primarily run Kotlin and Java; IOS can run Swift and Objective C.  The Jetbrains IDE seems best for Kotlin, Eclipse can work for Java, and Xcode will work for Swift and Objective C.  Two additional development teams will be needed, one to continue support for the android OS and another for the IOS development. |

## 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 believe that a very cost-effective solution would be to utilize a cloud-based hosting service which runs an enterprise distribution of Linux such as SUSE, Red Hat Linux, or similar. An advantage of this approach in that it reduces start-up costs and abstracts the costs of server maintenance. Another advantage is that this solution should be scalable in order to meet increased or reduced demand. Another advantage of this would be potentially enhanced server uptime, as contracting with a service which specializes in hosting will likely be able to offer a higher level of expertise in server maintenance than can be provided locally. A drawback to this approach might be a slightly increased cost of hosting over the long term; if demand remains stable and the system remains unchanged for many years it could be less expensive to purchase hardware and physically host it, but it could take a decade to gain a significant return on investment. A business decision to migrate to physical servers could be made at a later date.
2. **Operating Systems Architectures**: As mentioned above, the system will be a cloud-based solution. The system should be distributed into discrete systems which communicate via an API; Ideally, there would be a separate system for authentication and user data, for image library hosting, and for backend game logic. The server logic will be concerned with issues such as matchmaking and event timing; logic handled by clients will be image processing and user input.
3. **Storage Management**: As of right now the storage requirements which were identified seem rather modest; it is possible that this minor storage space may come with the cloud service that is used to host the main system. Otherwise, a cloud service specifically to host the image library, then may be connected with the cloud-based server or, potentially, have client-side software interface with the image library directly in accordance with directions sent by the cloud-based server. Having clients interface with the image library host directly is an increase in complexity, but it may save on cost and performance since the image would be sent to clients directly rather than routed through the server.
4. **Memory Management**: There may not be enough space in this paper to fully explain how memory management techniques work, however the system will use such methods as dynamic loading, which tries not to load a process until it is time to use it, swapping, which offloads processes to a larger and slower disk in order to free up space in faster areas, and paging, which is similar in concept to swapping, but for parts of a process rather than the process as a whole. I would recommend offloading rendering logic onto client-side applications and limiting the server to the roles of match-making and event-coordination. For example, before the match starts, I would advise the link to the image file be sent to the clients, have the individual clients download the image from the image library, translate it to the individual device’s resolution, then slice the image up into .25% lines, then signal the server that the client is ready to start the match. The server would start the game once all players are ready and send commands at appropriate times such as “display first .25%” and “display next .25%”. In this way, the server’s memory usage is kept at a minimum and the app utilizes the resources of the client device.
5. **Distributed Systems and Networks**: As outlined in other sections, I recommend that the server be focused primarily on event coordination: There needs to be an authentication process, a system which handles user information, a game creation process, a team creation process, and a game process. The game process on the server side can be simplified to coordinating events at specific times: It sends the image (or information about how to download the image), then sends instructions about when to display a certain portion of that image. All of this can be accomplished through an API. The clients will read the information on that API and use that information to handle and display menus and image rendering. Depending on the scale of the project, separate servers for authentication and hosting the image library can be used. More servers can potentially introduce more possibility of server outages, but is cloud-based solutions were employed as I recommended, I anticipate outages should be minimal. Adding a separate image library server would mean that the library would interface with the server through an API and potentially interface with clients directly through another API (in the event that the server would send download instructions to the client, then the client would request to download the image from the library).
6. **Security**: The scope of data collected from users should be limited; if there is no need to store a user’s phone number, then no phone number should be collected. When designing client software, care should be taken to ensure that the application is sufficiently isolated from other portions of the operating system and that malicious third-party software cannot easily harvest saved data. When transmitting information between the client and server, the communication method should be encrypted and the endpoints sufficiently constrained (via API or some other type of strictly defined communication style.) The surface area of the server should be minimized so that there are minimal areas of potential attack. Data should be encrypted and, if feasible, separate and specialized authentication servers ought to be employed. The permissions of various user roles within the server network should be regularly reviewed as well as ensuring that legacy systems are appropriately decommissioned. During such reviews, the firewall should be reviewed to ensure that there are no superfluous holes. The network should be monitored both by automated tools and by cyber security professionals; suspicious network activity such as large file transfers, atypical transfers outside of business hours, and other unusual requests. While everyone should at least consider adopting multifactor authentication, players may be resistant to added effort to log-in. Multi-factor authentication for employees should be mandatory. Employees should regularly receive cyber security refreshers such as how to spot phishing attempts.