School of Informatics, Computing & Cyber Systems

Technological Feasibility Analysis

Serpent Studios

November 9, 2022

Sponsor:
Dr. Patrick Kelley

Mentor:
Italo Santos

Team Members:
David Hermann
Johnathan Ray
Tyler Morales
Nickolas Maxwell
Nick Shugrue

Table of Contents

1 Introduction

Large companies such as Electronic Arts, Activision-Blizzard, and Bethesda with multi-million dollar franchises such as *FIFA*, *Diablo*, and *Fallout* respectively dominate the gaming industry. These studios produce games en masse, and as such ship products that are lackluster and stale. For example, a new *FIFA* game is released yearly but updates mostly consist of new characters based on roster changes made to teams registered with the International Federation of Association Football (FIFA). Recently, there has been an uptick in the gaming community's interest in original projects, new versions of older genres, or games that have been recreated from older works. A notable example is the roguelike genre renaissance which includes titles such as *The Binding of Isaac*, *FTL: Faster Than Light,* and *Hades*. This genre derives its name from *Rogue*, which was released in 1980. It features procedurally generated environments and permadeath, where losing a character means restarting a playthrough. New games often have innovative features either due to their rejection of conventional game loops or by improving upon previous design paradigms with new technology. Developments include engaging new stories which may be adapted from previous works of fiction that evoke nostalgia from older players while allowing a new generation of players to experience classic works in a fresh medium.

Large game studios inherit the resources of their parent companies whereas smaller independent developers cannot afford such tools. Previously, this would give larger studios an advantage. However, several smaller and more affordable "game engines", software packages with a large set of features for game development, have been made freely available to independent developers. In fact, some larger studios have even adapted these game engines for their titles, demonstrating how powerful such engines are.

Our sponsor, Patrick Kelley, was fond of a game found in *Serpent's Reach*, a novel he read when he was younger. He wanted to know if the game was something that could be played in the real world. He set out to use his programming abilities to bring the game to life digitally. Our client decided to use the game engine Unity, and made a basic version of the game based on rules proved by C. J. Cherryh, the author. This prototype has simplistic two dimensional visuals and an ambiguous user interface as to what is happening in the current game. Additionally, it does not contain features such as persistent scoring or peer to peer multiplayer. The prototype's code was written with only the developer in mind and as such it is inextensible; no additions can be made without rewriting the entire codebase. While Sej has a score system for tallying points, it does not work in the prototype. As for multiplayer, the current implementation requires two players to share one machine and take turns inputting their turns in order, which leads to possible confusion and errors as to whose turn it is.

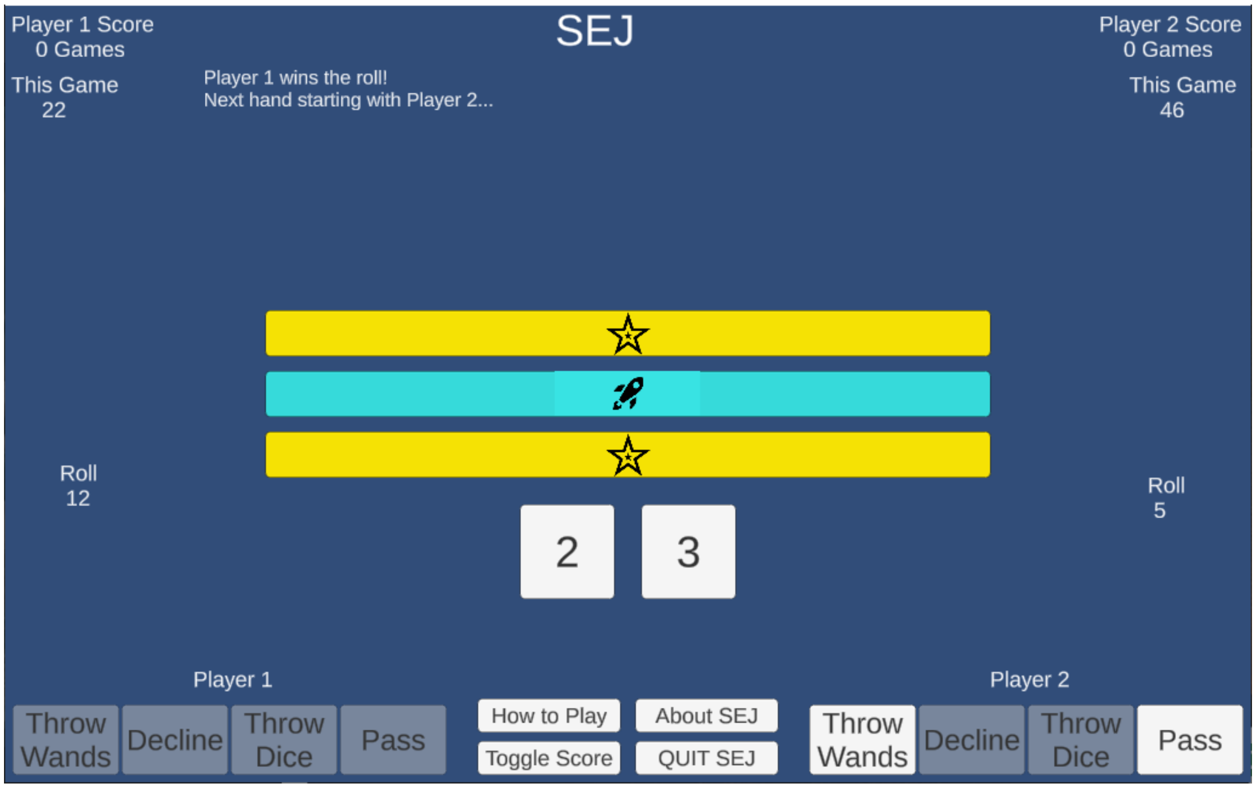

Figure 1.1: A screenshot of Patrick Kelley's Sej prototype.

Having done research into pertinent technologies this document expands upon the technological feasibility of Sej Online. Our team will implement key professional studio practices, techniques, technologies, challenges, and solutions that we believe will take our small independent project to the next level, capable of competition with larger development studios. By using Unity, a game engine known for its feature-rich codebase and extensive third-party support, we intend to implement persistent user scoring, a better user interface, online multiplayer and improved graphics for the minimum viable product. To solve the problem of persistent scoring, Database Control acts as a script that translates C code into instructions on what data to store in a database. This data can be retrieved later so that players can see previous scores and statistics. The planned user interface will include a better dice and wand rolling system, responsive button input, and cleaner visual clarity to avoid ambiguity. Since local multiplayer using a single machine is not scalable, we will utilize engine related plugins, already developed and in use by gaming studios to provide a networked multiplayer solution to our players. Finally, Unity provides many libraries and tools for advanced graphics, both in two-dimensional and three-dimensional formats. Our implementation will provide cleaner visuals and possibly some advanced graphics such as three-dimensional dice and wands.

This document discusses the technological challenges we expect to face, an analysis of available technologies and why Unity was chosen as the game engine, and how we intend to integrate Unity and other technologies into our development cycle. The technological challenges of Sej Online include the usage of a game engine, the design and implementation of an improved user interface, a database for user statistics and score tallying, network connected multiplayer, and animations and graphics. In our research, multiple possible technologies were found that meet the needs of the minimum viable product. The technology analysis section discusses candidates for each of the features mentioned in the technology challenges section and discusses the desired characteristics, alternatives, and chosen approach for each feature. Finally, the technological integration section discusses how all of the features will be implemented in conjunction with Unity.

2 Technological Challenges

When creating an online game, each possible feature requires thorough research, discussion and revision. This section identifies and discusses many of the potential challenges that may arise once development of Sej Online begins. First, an appropriate game engine must be chosen to act as a framework for the game. There are many great engines to choose from, and the features and drawbacks of several will be discussed in the technology analysis section. To guide that discussion, factors to consider include the ability to create an intuitive and interesting user interface, animation and graphical display capabilities, being able to transfer data to and from an external database, and support for network connected online play. Below is a basic outline of the expected Sej Online architecture.

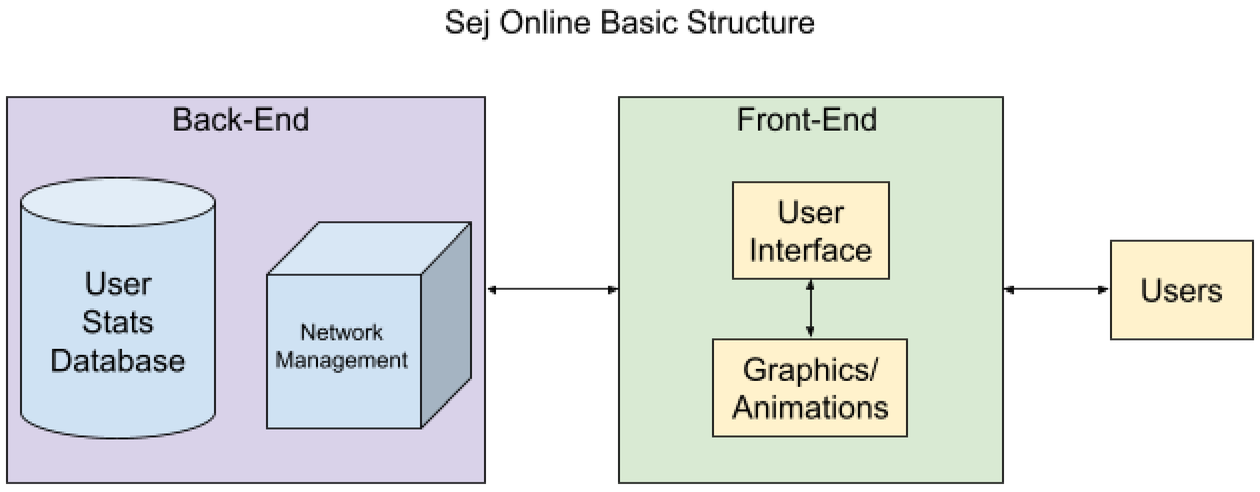

Figure 2.1: A simplified Sej Online structure.

The key elements needed to achieve our envisioned solution for Sej Online are as follows:

- **Game Engine** - The first and most important decision that must be made before development can begin. Game engines provide a useful framework for game development, offering helpful libraries and support programs. The benefits and drawbacks of several are discussed in the technology analysis section.
- **User Interface** - This can make or break the entire user experience for any game. An intuitive and interesting UI is key to keeping users engaged. A poor UI can make a game

feel lifeless and hard to navigate. The Sej Online UI will contain both a menu system and a play window while the game is running.

- **User Statistics Database** - Any and all user statistics will be cumulatively stored in some sort of database. This adds a sort of progression that can incentivize users to continue playing the game, as well as creates a competitive aspect to the game. Users will be able to access and compare their states with other users.
- **Network Connected Two-Player Mode** - Sej Online will allow users to play against other users on different machines. This will create a community around the game and allow users to interact with each other without having to be in the same room.
- **Animations and Graphics** - These can come in a variety of forms both simple and complex. Animations and graphics help make the game more interesting to look at. They can also keep the user interested for longer sessions of play.

The following subsections introduce each challenge and qualify the important aspects to consider.

2.1 Game Engine

Sej Online will be developed with the assistance of a game engine. Game engines provide an important framework for developing games. They offer tons of features and resources for developers to create all kinds of games. Many engines have dozens of libraries and supporting programs that aid in making the development process more efficient. Ideally, the game engine we choose to go with will give us all the tools we need to overcome the challenges we will face while developing the game.

Most game engines allow developers to add the following features:

- Physics
- Input
- Rendering
- Scripting
- Collision Detection

- Artificial Intelligence
- And more, without the need to program them from scratch

For our purposes, we will not need most of these features until the minimum viable product is complete. The main feature necessary for Sej Online is multiplayer support within the engine. Since the project will be an online game, we will need to implement multiplayer functionality, and having those tools built into the engine itself will make that task much easier. It will also be important to look at the practical features of the engine for quality of life and ease of use purposes. The aspects we will be looking at are:

- **Pricing** - How much does it cost to use the engine?
- **Learning Curve** - How long will it take to become familiar with the engine?
- **Languages** - Which programming languages does the engine use?

The engine will hopefully give us a good balance between functionality and efficiency.

2.2 User Interface

A simple but attractive user interface (UI) will be necessary for both the menu and play window of the game. The UI will have to be intuitive and functional while also being interesting to look at for the user. Creating a good looking UI that does everything we need it to may be challenging without a significant background in design work. Luckily, there are several common elements that make up a successful UI:

- **Simplicity** - The best interfaces avoid unnecessary elements and are clearly labeled.
- **Consistency** - Creating consistent patterns in the layout and design can facilitate efficiency. Users should be able to transfer knowledge between different parts of the game.
- **Purpose** - Careful placement of items can help draw attention to the most important pieces of information and can improve readability. It is important to consider the spatial relationships between items, and the structure should be based on importance.

- **Strategy** - Color and texture direct attention toward or away from items.
- **Typography** - Careful consideration of the typefaces used can increase legibility and readability. Use different sizes, fonts, and arrangements.
- **Communication** - Always inform users of any changes, actions, or errors in order to reduce confusion and frustration.

Using these UI best practices, we can develop a user interface that is pleasant to use and easy to understand.

2.3 User Statistics Database

Sej Online will utilize a database management system (DBMS) in order to store gameplay statistics for its users. A DBMS is software that manages the storage, retrieval, and updating of data. Most databases store this data in some sort of table to make it easy to understand how the data relates to each other. A database would be able to retrieve and compare different data sets for user analysis. For example, users should be able to see how many games they have played, won, and lost in total, as well as against a specific opponent. The database used for Sej Online should be able to accomplish the following:

- **Create secured accounts for users** - Players will have an account linked to their machine that tracks any and all progress made. This will include statistics such as games played, wins, losses, etc.
- **Compare statistics between users** - Players will be able to compare their own stats with those of other players. They will also be able to access stats against specific opponents.
- **Display relevant statistics to all users** - It might be interesting to create a type of global leaderboard that shows all time statistics for the best players globally. This would require the ability to rank players by their statistics and display that information to all users through the game menu.

If a DBMS is to fulfill the above requirements, it must have the following characteristics:

- **Maintainability** - The DBMS should be easily maintained and managed. It should not be hard or confusing for changes or improvements to be made. It also should not need frequent changes to function properly. It should continue working consistently with little downtime. It is possible that users could be playing the game at any time, so the database should not be hard to keep running otherwise a core mechanic of the game could become unavailable.
- **Extensibility** - It does not make sense for there to be any limit on the number of users that can create accounts for the game. The database should be able to handle an increasing number of player accounts for the foreseeable future. It should also be able to handle new game features that require database access beyond the initial scope of the original product.
- **Accessibility** - Since the database will be used for several aspects of the user interface and gameplay, it should not be hard to access the necessary data. It should be easy to access and update user data every time a Sej match is completed. It should also be trivial to retrieve user data every time a user looks at their statistics on their profile.

A viable DBMS will meet or exceed the characteristics outlined above.

2.4 Network Connected Two-Player Mode

Sej Online will obviously be an online game, so networked, multiplayer supported gameplay is required. This is a key factor in creating an interesting and interactive game, so getting it right is vital to the success of the product. Unfortunately, perfect online play does not exist. Even the most modern and advanced online games still fall victim to poor, unstable, or lost connections. There are many issues to consider when creating an online game. Some of the major issues include:

- **Consistency** - A consistent connection between users will be vital to the gameplay experience of Sej Online. Lag, jitter, dropped packets, lost connections, etc. can be massively detrimental to the success of the game.

- **Reliability** - Online play should be accessible at any and all times of the day or night. Users should be able to play whenever they want for as long as they want without having to worry about network issues.
- **Latency** - It should not take hours or even minutes for actions to be transmitted from one user to another. There should be very little delay for any process in the game, otherwise users may believe the game has stopped working.

2.5 Animations and Graphics

The most successful games are ones that keep the user wanting to come back for more. A great way to keep users interested and engaged is with appealing graphics and lively animations. These do not have to be amazing or groundbreaking; they should be enough to impress new and returning players. Good graphics and animations may include some of the following elements:

- **Simple** - It is easy to overdo it with super flashy visuals, but it is important not to overstimulate the user, especially in the case of photosensitive individuals. Simple is better in this case.
- **Informative** - Visuals should be informational and signal some sort of action or event to the user. This creates a more engaging and understandable gameplay experience.
- **Relevant** - An out of place visual ruins immersion and engagement when playing any game. It is important to make sure all visuals fit into the theme of the game so nothing feels out of place.
- **High Quality** - Low quality graphics or animations can be a distracting and unpleasing sight. Visuals should be of good quality and add to the experience, rather than take away.

Good visuals can aid in creating an enjoyable experience and atmosphere for the user. Any graphics or animations used should follow the guidelines above.

3 Technology Analysis

As stated previously, the biggest challenge that we must address is choosing a game engine. The engine needs to allow us to solve four challenges: ensuring that two users can play with each other on separate machines, allowing users to create an account that will record their wins and losses, and creating an intuitive UI and appealing graphics. After researching the wide variety of engines that are available, we found that Unity, Unreal, and Godot were the best three options in accomplishing these tasks. The table below provides an overview of each of the challenges and the pros and cons of how the engine can solve them. This will be discussed further in the rest of this section.

Engine	Programming Languages	Supported platforms	UI; Graphics	Multiplayer support	Database support
Unity	C#	All	UI toolkit, Unity interface package, IMGUI; high-end graphical capabilities	Built-in library (Netcode), 3rd party (Mirror)	Database Control
Unreal	C++, Blueprint	All	Widgen Blueprint; high-end graphical capabilities	Built-in library	N/A
GoDot	GDScript, C++,C#	All	Built-in tools; 2D graphical capabilities	Built-in library	N/A

Figure 3.1: An overview of how each engine can be used to solve the four challenges.

3.1 Game Engines

Each part of the project is important, but the game engine serves as the foundation of creating our project. If the team chooses the wrong one to create Sej, then the entire project is unfeasible from the start. It needs to allow us to implement the game's ruleset without too much difficulty. The difficulties that we may encounter are expanded upon in the following sections.

- **Desired Characteristics -** When selecting an engine, we first need a tool that does not take very long to set up on our computers. That is, we need an engine that is free and takes up a reasonable amount of storage space so that it can actually be used and will allow us to make a game that does not take up much more additional space. Once that is found, we then need an engine that will allow us to create a game like Sej (that is, a tabletop game that involves making strategic decisions based on randomly generated scenarios). It also needs to be able to support all major platforms, including console, mobile, PC, and Mac.
- **Alternatives -** The three engines that we are considering are Unity, Unreal, and Godot. Unity first released in 2005, and the earliest long term support (LTS) build came out in 2020. Unity is considered to be a "jack of all trades", which can be exemplified by the wide variety of games made in the engine, including *Among Us*, *Monument Valley*, and *Hearthstone*. It uses C# as its scripting language and supports all major platforms.

  Unreal first released in 1998 by Epic Games. Many modern games that have used Unreal used Unreal Engine 4, a version that was released in 2014. This year, however, Epic released a newer version, Unreal Engine 5, that is slowly being picked up by developers in the industry. The majority of games made with Unreal are 3D and have high-end graphics. It uses C++ as its scripting language, as well as Blueprint, a built-in language that uses a click-and-drag system to provide functionality to different objects in a game. It also supports all major platforms

  Godot first released in 2014, and the current version was released in 2022. While it can make 2D and 3D games, the majority of games are primarily 2D. Unlike Unity and Unreal, Godot organizes the objects in a game as nodes in a tree. It uses C# and C++ as its scripting languages, as well as GDScript, a built-in language that is similar to Python in syntax. It supports all major platforms except for consoles.
- **Analysis -** As most of our team has experience working with Unity, we understand its ins and outs fairly well. The base LTS build released in 2020 that we installed weighs in at around seven gigabytes and is free to use. According to our research, the other LTS builds are of similar size. This allows us to install other tools and assets without having to worry

about freeing more space. As stated previously, the engine is a “jack of all trades”, meaning that it allows one to make any type of game they want. When testing this claim ourselves, we found that one could select from a list of prebuilt level templates (e.g. third-person, top-down, etc.). Unity also allows developers to port their games to all major platforms. When testing this, we found that this can easily be done by selecting a platform from a provided list and saving the files to a zip file.

Unreal Engine 5 is the biggest engine of the three, with its free base version weighing in at around fifty-five gigabytes. Even without taking into account the other tools and assets that we may want to use, the amount of storage this takes up is already very large. When making our own test projects in Unreal, we found that it had many similarities to Unity: it allows a developer to choose from a selection of template levels which can be ported to any major platform by choosing from a list provided by Epic. The graphic capabilities of the engine are much greater than that of Unity. Making a version of Sej with graphics of such a high caliber, however, is beyond the scope of this project. Even if the team wanted to do so, we found that loading in shaders from example projects provided by Epic can take extended periods of time to load in and may even result in a system crash.

Godot is the smallest of the three engines, weighing in at around 500 megabytes. Despite its size, however, it still provides a development environment similar to that of Unity and Unreal. Its graphical capabilities are similar to those of Unity as well. Unlike the two engines, however, Godot does not provide a selection of template levels for a developer to choose from. Additionally, it does not port to consoles. It is also the black sheep of the three due to it using GDScript, its own scripting language. While it is Pythonic in its syntax, having to learn another language may not be feasible, given the amount of time we have to develop this project.

- **Chosen approach -** Each of the three game engines has its own pros and cons. All of them are free and use languages that are familiar to all of us (e.g. C#, C++), but they also have their unique nuances. Some of them allow a developer to make a specific type of game, while others are more flexible. Some are large in size, while others are fairly small. Some port to all platforms, while others port to some. Figure 3.2 lists all the pros and cons of the engines in terms of desired characteristics listed previously, as well as a rating based on our analysis.

	Storage Size	Types of Games	Programming Languages	Supported Platforms	Overall Rating
Unity	7 GB (4/5)	Any (5/5)	C# (5/5)	Console, mobile, PC, Mac (5/5)	4.5/5
Unreal	55 GB (2/5)	3D Games with high-end graphics (3/5)	C++ (4/5)	Console, mobile, PC, Mac (5/5)	3/5
Godot	500 MB (5/5)	2D Games (3/5)	C#, C++, GDScript (4/5)	Mobile, PC, Mac (2/5)	3.5/5

Figure 3.2: An overview of how each engine contributes to the desired characteristics listed in each column, along with a rating for each characteristic.

After analyzing the benefits and flaws of each of the engines, we believe that Unity seems to be the best fit for our project. Our group is familiar with both the engine itself and C#. Not only that, but it fulfills all of the desired characteristics that we need in a game engine. Its storage size is not the smallest, but it is small enough that we can make our game without having to worry about it taking up too much space alongside Unity. It is also a versatile engine that can make any game we want, which is desirable for the specific type of game we want to make. Finally, it supports all major platforms and allows developers to easily choose which ones they want to port their game to.

- **Proving feasibility -** Our client has already demonstrated that implementing the rules of Sej is feasible. In our future tests with Unity, we will ensure that this is so and expand on the initial concept that was presented to us. For instance, we will create 2D and 3D builds of Sej to see which medium is a better fit for the game. This will be determined by analyzing how much space it takes up in storage, the simplicity/complexity of the two mediums, and which one is more engaging to play.

3.2 UI & Graphics

The initial prototype shown by our client does not have the best graphics. In short, they are unengaging and do a poor job at providing the necessary information needed to play Sej. While this is an aspect that we believe we can improve upon, an aspect that we must improve upon is the user interface. The client's prototype's UI is clunky and also unengaging. In order to create a top-notch version of this game, both the UI and graphics need to be overhauled in a significant way.

- **Desired Characteristics -** At first, it may seem that the UI and graphics are two separate challenges. As seen in the previous section, however, they have many similarities. They both need to be simple and consistent enough so that the user can quickly understand how Sej works, even before understanding the full ruleset, and is not confused by all the information on the screen. The user interface specifically needs to contain information that has a purpose, so an engine should allow for a UI element to be customizable enough that our game can communicate this information to the user. The graphics specifically need to be high quality, allowing the player to play the game without being displeased by what they are looking at. An engine should be capable of producing graphics that allows for this. In addition, it should allow for a developer to be flexible with how their game's graphics look in the event that the game starts to take up too much space.
- **Alternatives -** Much like Unity itself, the graphical capabilities of the engine are flexible. One can make a game with high-end or low-end graphics. Unity also has several systems for developing UI elements, including the UI Toolkit, the Unity user interface package, and Immediate Mode GUI (IMGUI).

  Unreal is one of the best engines out there for making games with high-end graphics. Specifically, Unreal Engine 5 has implemented new technologies that allow for developers to make some of the best looking games on the market. That being said, it can still be used to create games that do not demand high graphical quality. Additionally, Unreal uses a tool called a Widget Blueprint that uses the engine's built-in scripting language to create a user interface.

As stated previously, Godot is primarily used to make 2D games, so its graphical capabilities are designed with that in mind. The engine's UI tools are integrated with the rest of its tree structure elements, much like Godot's other tools.

- **Analysis -** Considering the user interface and graphics have so much in common it is only natural that they would share applications for their development. In our research we discovered several tools that will help in our approach to creating visuals for both avenues that maintain the foundations of simplicity, consistency, that contain relevant information and are informative, and finally that are of high quality.

  For creating user interfaces unity offers three distinct systems, UI Toolkit, the newest system and well optimized. It can provide runtime UI for games built within the engine. Second, is the unity user interface package, allowing further runtime user interfaces to be built within the engine itself. Finally, IMGUI which is designed primarily for aiding in debugging and editing the engine interface. Unreal has a tool called the Widget Blueprint that can create robust UI elements. While it does provide a good amount of flexibility in creating simple and purposeful user interfaces, it does not provide near the amount that Unity has. Godot's UI tools function similarly to Unreal's. Again, though, it offers nowhere near the amount of flexibility that Unity offers. Despite this, both Unreal and Godot's tools would provide the flexibility necessary to design the UI we had in mind.

  Each of these UI tools are great, but we still need to find a solution with better graphic capabilities. There are options such as Blender, a great program for making 3D assets. Unfortunately this may be outside of the scope for our project, as we intend to focus first on two-dimensional graphics. Unreal offers immensely strong graphics capabilities, but again, this engine is designed to take on much larger tasks. Godot offers a limited amount of graphical capabilities that can still deliver in terms of simplicity and quality.

  One such example of a simpler solution is the program Aseprite, a pixel drawing and animation tool that is very simple and easy to approach. The software has a multitude of features for animating and creating great looking interface designs and integrates well with Unity. It even has several plugins that help to facilitate the importing and exporting between the two pieces of software. Combining the Unity game engine built in user interface tools, in unison with Aseprite we are capable of avoiding overly complex asset creation tools, while still maintaining professional level quality.

	User Interface Capability and Flexibility	Graphical Quality and Capability	Overall Rating
Unity	Has three different systems for producing UI elements. Each one provides a great amount of flexibility. (5/5)	Capable of producing high- and low-end graphics, allowing developers to be flexible and create simple and easy to understand visuals. (5/5)	5/5
Unreal	Has built-in tools that allow for the creation of UI elements; provide an average amount of flexibility in terms of creating these elements. (4/5)	Produces some of the highest quality graphics in the industry, but this may be beyond the scope of the team's vision (4/5)	4/5
Godot	Has built-in tools that provide an average amount of flexibility when creating UI elements. (4/5)	Has the capability of producing high-quality visuals; on par with Unity. (5/5)	4.5/5

Figure 3.3: An overview of what each engine's graphical and UI capabilities can contribute to the desired characteristics, along with ratings for each one.

- **Chosen approach -** Figure 3.3 shows that, when all is said and done, each engine's graphic and UI tools can get the job done. There are some, however, that are capable of achieving the desired characteristics better than others. The flexibility of each of these tools plays a large role in how they can overcome this challenge. After analyzing all of these tools, we reached the conclusion that Unity's graphic and user interface are the best fit for our project. The flexibility that its UI tools provide will allow us to create a simple interface that allows the user to easily understand the purpose of everything that they are taking in. Additionally, the graphical capabilities of Unity allow to create visuals that are high quality while, at the same time, are simple enough to assist the player in understanding how the game works. On top of that, if we want to improve the visuals in any way, the tools are easily scalable and can allow us to do so.

- **Proving feasibility -** In choosing tools that are already used at the industry level for user interface, graphics, animation, and all other assets, the studio ensures the development of a quality product. Pre-built industry standard tools avoids the need to worry about whether the tools selected will allow our artists to do what they need, and rather allows them to simply focus on creating quality assets, when concerning the development of user interfaces, and graphics.

3.3 Support Online Two-Player Mode

While the build of Sej provided by the client includes a local two-player mode, it does not include an online multiplayer mode. At a minimum, an online version of this game should do the same things as an offline version: two players should be able to play a standard game of Sej without any interruptions or pauses (apart from those made by the players). In order to accomplish this, we need a networking library that can fulfill two desired characteristics, which are explained below.

- **Desired Characteristics -** Since Sej is a multiplayer game by design, a user should have the ability to find other people to play with easily. Otherwise, the game is pointless. Additionally, when they find someone to play with, the two players should not have to wait an extended period of time to get started with their game. Finding a server to play on should take the same amount of time as setting up a game of Sej in real life. With this in mind, we believe that, when a user is ready to play a game of Sej, they should be able to search for another user and join a server in a short period of time.
- **Alternatives -** Unity has a library called Netcode that was released on June 27, 2022. This built-in tool was created by the Unity team and supports all LTS versions of Unity from 2020 on. As this was recently released, there are not many developers that have used this in their games. In fact, many of them used Unity's previous networking library: UNet. According to its documentation, however, Netcode has many similarities to Unet

and allows developers to smoothly transition between the two libraries. In addition to Netcode, Unity also has numerous third party libraries, such as Mirror and Photon, that are similar to Netcode and its predecessor and have been supported for longer periods of time.

Unreal has a built-in library that has been supported since 2014. The documentation provides an extensive exploration of what a developer needs to do in order to make a multiplayer network. Additionally, unlike Unity and Godot, it delves into how one can create a dedicated server. This would allow a session between two users to run smoothly without much wait time during the game.

Godot offers a multiplayer API that, by default, provides implementations based on ENet, WebRTC, or WebSocket. Additionally, it provides higher level facilities that one can use if they do not want/need to deal with the specifics of the previously mentioned network implementations.

- **Analysis -** As stated previously, Netcode has a descriptive documentation page that clearly states what needs to be done in order to get a multiplayer server up and running. Additionally, there are numerous video tutorials that clearly explain how this library works. After viewing these two resources, the team created a small Unity project that utilized the tools provided in Netcode and were able to create a server without too much difficulty. The server we made was responsive and did not have any delays on either side. In the event that delays occur, however, the documentation provides advice on how to handle this. There are also several third-party plug-ins that allow for quality of life improvements when working with Netcode. ParrelSync, for instance, allows developers to test the network without having to rebuild the game every time.

  Overall, Netcode fulfills all the desired characteristics. Again, though, Netcode has only been publicly available for a few months, so there are not too many resources that provide explanations for any hiccups that the team will inevitably run into. Not only that, but the documentation states that it does not support WebGL, meaning that we could not run the game in a web browser. For these reasons, it may be beneficial to use the Mirror library in the event that Netcode is not the right fit for Sej. While the team has not yet made a project to test its capabilities, it is our understanding that Mirror has a similar functionality to Netcode.

The team has also yet to make a project that tests Unreal and Godot's networking libraries. Upon reading the documentation for both, however, we believe that they could still be good fits for creating Sej, albeit with their own shortcomings. For instance, Unreal Engine 5 shares many similarities with Unreal Engine 4. Because of this, we believe that Unreal's networking tools have been supported since Unreal Engine 4's release in 2014. Simply put, the library is widely used and there are plenty of resources that can provide information on how it works and how it can go wrong. The documentation describes how a server should be set up and provides advice for how to smoothly connect two players and keep that connection up and running. However, it does not provide a succinct explanation for how two players can find each other, so it is unknown how quickly two players can find each other. It also describes how one can design a dedicated server for their multiplayer game, which we may want to use to minimize wait time for the two users. Other resources state that running a dedicated server may cost money, however, which we want to avoid if at all possible.

Godot's networking library, much like Unreal, has also been supported for an extended period of time. This means that there are also numerous resources explaining how it works and what can go wrong when working with it. It provides a high-level API for developers who do not want to mess with the specifics of the network sockets. That being said, it also has a mid-level API for those who want to change the specifics for the game they have in mind. If we were to use this library, we would most likely utilize the high-level API. While it may be useful in the event that we need to deal with the minute details of a network, using the mid-level API is beyond the scope of the project and has the potential to get in the way of development. Apart from that, the library seems to be able to find and connect two users quickly and keep the server running without any wait time on either end.

	Networking Library	Finding Other Players	Wait Time/Lag	Shortcomings	Overall Rating
Unity	Built-in library (Netcode), 3rd party (Mirror)	In our tests, we found that connecting two players was simple and fast. (5/5)	None (5/5)	Netcode is new and does not have many resources that show how things can go wrong. (2/5)	4/5
Unreal	Built-in library	The documentation provides a succinct explanation for setting up a server, but it does not state how quickly two players can find each other (3/5)	None (based on our research into running dedicated server) (5/5)	Running a dedicated server in Unreal may cost money. (1/5)	3/5
Godot	Built-in library	Based on our research, we found that setting up a server with Godot is simple and does not take too much time. (5/5)	None (based on our research) (5/5)	The mid-level API is beyond the scope of our project and may get in the way of development. (2/5)	4/5

Figure 3.4: An overview of how each engine's networking library contributes to the desired characteristics listed in each column, along with a rating.

- **Chosen approach -** Including a built-in library for networking tools is very helpful for the average developer, and it is amazing that all three engines have this feature. Not only that, but they all run fairly smoothly and have little to no lag between users. However, they still have their shortcomings. Figure 3.4 lists all the pros and cons each of the networking libraries provides in terms of the desired characteristics listed previously, as well as a rating based on our analysis.

  After analyzing the benefits and flaws of each of the libraries, it was a toss-up between Netcode and Godot's built-in library. However, based on the tools we want to use for our other challenges, we believe that Netcode is the better choice. First and foremost, it satisfies the desired characteristics of a multiplayer network in our game, one that allows two users to easily find each other and stay connected without there being too much lag. The biggest downside to using this library is the fact that there are few resources that explain how things can go wrong. Because of this, we may find that it is not a great fit for Sej. In the event that this happens, we can always switch to Mirror. It has a similar functionality to Netcode, but it has been supported for a longer period of time. This should smooth out the process of troubleshooting in the event that we run into problems while working with it.
- **Proving feasibility -** While we have already run tests to learn how Netcode works, we do not know how well it works with games like Sej. Because of this, we will develop a test build that demonstrates that Netcode can run Sej. The test will include a barebones version of the game that includes the core mechanics such as the dice, wands, and player score, and none of the extra material such as a copy of the rules or high-end graphics.

3.4 Create User Account

The build we received unfortunately had no capabilities for persistent scoring. Today's games have user data that far exceeds just that of persistent scoring. Our game solves this by creating user accounts, with scores, games played including those won and lost. This data will be offered in a fashion such that players may refer to games won or lost against other players. There are many ways to attempt this, and we have found the approach of a lightweight scalable database to be the best fit for our user data. This allows us a way to keep all of the data safe and secure, and readily available for players.

- **Desired Characteristics -** The database for Sej does not require too much information from a given user; all it needs is a username and a password. Given that the latter must be confidential, however, the database should provide some level of security. Once the user creates their account, it should record and provide their wins, losses, and other information that they may find relevant.
- **Alternatives -** In the above section we mentioned the use of SQLite as a solution. In a situation needing more customizability, or possibly the design of new more advanced features requiring a new database solution, we believe SQLite in combination with the web programming language PHP will be the most appropriate solution as this will offer unparalleled customizability. In this alternative approach we will write the database as needed in SQLite using the C programming language. Then, in order to send and receive our data we will utilize PHP, where php acts as an interface to this database with the game client, thus providing a unique database solution.
- **Analysis -** In deciding to use the game engine Unity, we have been fortunate to find that there are a multitude of solutions for data storage. Traditionally in games a player database containing all pertinent information is written in a language such as MYSQL, hosted on a server somewhere, and finally the database is usually connected or interfaced with the game client via languages such as PHP.

  This method of approach is time consuming, and requires team capabilities in several areas. This can be difficult for smaller studios such as ours. One example of such a database solution that is used in the professional gaming industry is SQLite. SQLite is the most used database engine in the world. It is small, fast, fully capable of performing all tasks required of a database, and is highly reliable. Although, as stated, we do not need large amounts of information from a user, rather our user data primarily consists of simply a few strings and integers. This allows us a new approach to select from simple and lightweight plugins designed for the unity game engine instead of the larger overhaul of training for, and developing a complex database solution using tools such as SQLite.
- **Chosen approach -** Database Control (Unity plugin)**.** Our desired approach is to use the Unity plugin Database Control. The plugin is simply imported into the Unity game

engine workspace. From there the designer is granted access to a multitude of tools to create and integrate a database for any project. These tools include features to edit a database, such as empty, removing all users and data from the table. Database control provides features via a C# script to allow users to register accounts, these accounts are integrated with the database, and the user may then login using these newly recognized credentials. Database control offers the ability to set and get data using the same c script feature within the tool. The plugin offers further customization if the default script does not suffice, allowing developers to change the look and layout of the database interface to their liking.

Database Solutions:	Complexity	Standalone	Security
Database Control	Yes	All features within game engine	Very little(with Pro version)
SQLite	No	Requires use of web programming language to link client and database	Great Security Features. Requires some knowledge of SQL programming
MongoDB	Yes, but no current Unity support	Must be done through http requests made via game client	Security features can be implemented via code.

Figure 3.5: An overview of the different solutions researched, and their capabilities.

- **Proving feasibility -** In proving feasibility we can look into other projects created with the same tools for the efficacy of database control. When reviewing the product we have found several tutorials, and video tutorials demonstrating the product and its capabilities in action. The database control plugin is offered in the Unity asset store, and as such has met rigorous standards and quality control before being allowed as an offering. There are also several other developer accounts of using the software and their experiences. We know having a product that offers great support by both the developer and the community that uses it is a solution that will be in our favor.

4 Technology Integration

Building a video game requires intense detail to ensure that all of the moving parts are running smoothly. This section outlines how we plan to integrate and unify the technologies discussed in the technology analysis section into one solution. Since Unity was deemed the best candidate for game engine, this gives our architecture a solid base on how we will structure our solutions for implementing the rules of Sej, the user interface, database, and networking.

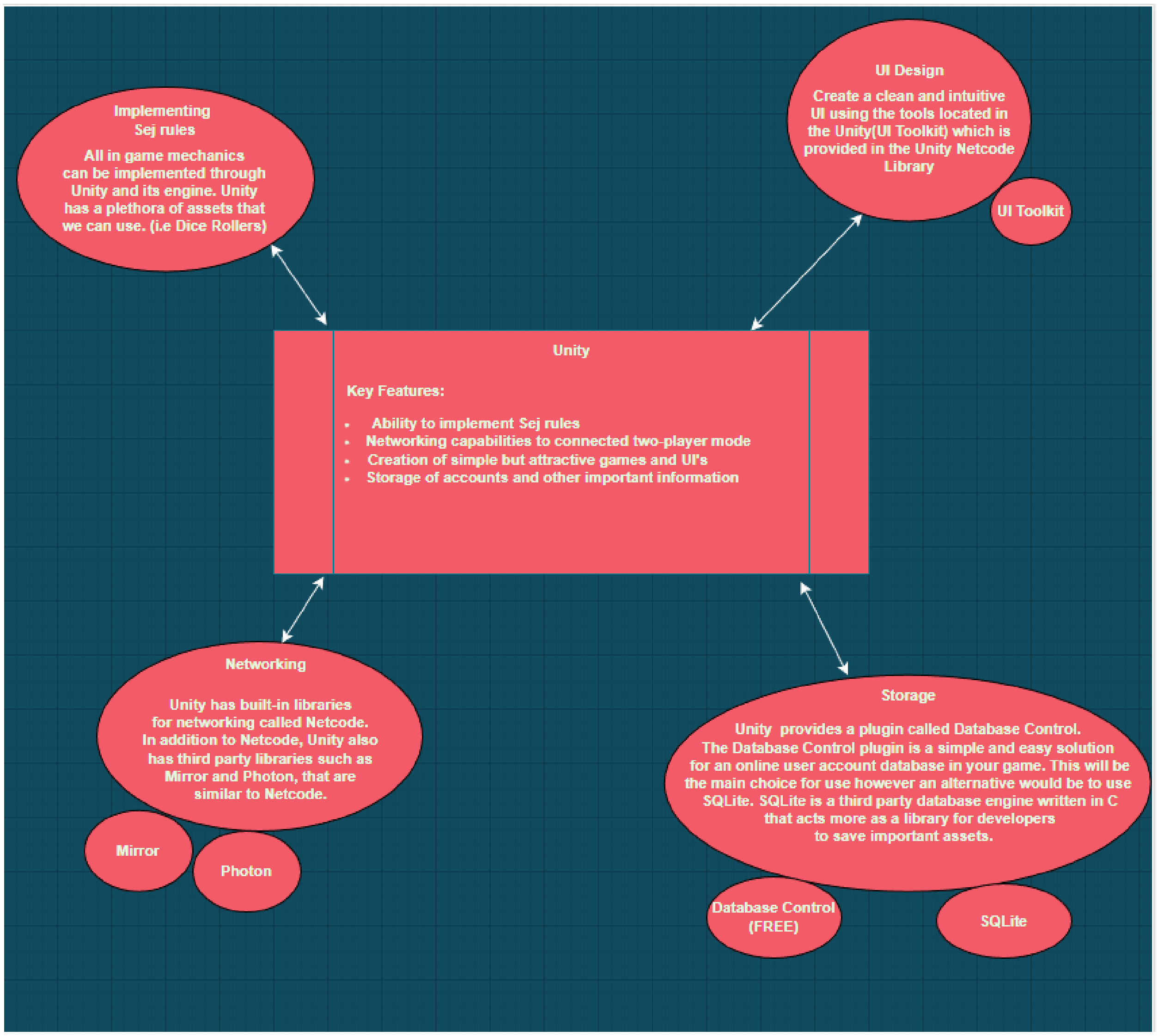
Implementing Sej rules
All in game mechanics can be implemented through Unity and its engine. Unity has a plethora of assets that we can use. (i.e Dice Rollers)
UI Design
Create a clean and intuitive UI using the tools located in the Unity(UI Toolkit) which is provided in the Unity Netcode Library
UI Toolkit
Unity
Key Features:
Ability to implement Sej rules
Networking capabilities to connected two-player mode
Creation of simple but attractive games and UI's
Storage of accounts and other important information
Networking
Unity has built-in libraries for networking called Netcode. In addition to Netcode, Unity also has third party libraries such as Mirror and Photon, that are similar to Netcode.
Mirror
Photon
Storage
Unity provides a plugin called Database Control. The Database Control plugin is a simple and easy solution for an online user account database in your game. This will be the main choice for use however an alternative would be to use SQLite. SQLite is a third party database engine written in C that acts more as a library for developers to save important assets.
Database Control (FREE)
SQLite

Figure 4.1: An overview of how Unity can contribute to each of our key features

The minimum viable product for Sej Online includes implementing the rules of Sej, setting up networking capabilities, creating an appealing UI/experience, and finally saving our users account information in a third party database through Unity. Unity is what joins everything together. Many of the key features that we will incorporate are tools that Unity provides. For example, implementing the Sej rules and game mechanics can be done in a variety of ways using Unity such as manually writing the necessary scripts; Unity also offers dice rolling assets. Unity further presents a plethora of choices when it comes to UI Design. Both two and three-dimensional graphics are offered as well as an extensive amount of tools (Visual tree, UI Renderer, Debugger) in the UI tool box. Netcode for GameObjects is also used by Unity to handle its networking. Unity describes Netcode as "a high-level networking library built for Unity for you to abstract networking logic." Netcode allows users to send GameObjects and other in-game data across a networking session to multiple players at once. Finally for data storage Unity has a plugin called Database Control. Unity describes the plugin as "a quick and easy solution for an online user account database in your game". This will be the main choice for use however an alternative would be to use SQLite. SQlite is a third party database engine written in C that acts more as a library for developers to save important assets. The integration of all of the technology required to create Sej Online was simplified with Unity. Unity brings everything together acting as a base for us to create this game as well as providing a multitude of useful resources.

5 Conclusion

To conclude, Sej Online seems technologically feasible and has potential to demonstrate small projects with vision can adapt from older works to deliver a unique experience in an industry pushed to deliver products with widespread appeal but bland experiences. Research into the technological challenges of what game engine to use, the design of the user interface, how to store user data, networking, and graphics has yielded useful tools that will come in handy in the implementation of Sej Online. Unity seems to be the engine with the best fit that has support for all of the requirements of the minimum viable product including the user interface and graphics, hooks into a database, and networking. Integrating these features into one product was proven plausible in the technological analysis and integration sections. Serpent Studios looks forward to using these technologies to implement a fully-featured and mature version of Sej that caters to both readers of Serpent's Reach and players unfamiliar with science fiction in the coming weeks.