



**MEDICAL GAMING SOLUTIONS**  
LEVELING UP HEALTHCARE THROUGH GAMING

# **Technological Feasibility**

**10-3-23**

**Project Sponsor: Dr. Ashish Amresh**

**Faculty Member: Italo Santos**

**Team Members:**

**Rain Bigsby, Veronica Cardenas,**

**Ethan Ikhifa, Lenin Valdivia**

**Overview:** The purpose of this team document is to outline the main technological challenges, explore existing products to utilize for development, and ultimately make a decision on technologies to pursue. This feasibility analysis will provide guidance on key design challenges.

# Table of Contents

---

<b>1.0 INTRODUCTION.....</b>	<b>2</b>
<b>2.0 TECHNOLOGICAL CHALLENGES.....</b>	<b>3</b>
<b>3.0 TECHNOLOGY ANALYSIS.....</b>	<b>4</b>
3.1 Software Reusability.....	4
3.1.1 Desired Characteristics.....	4
3.1.2 Alternative Design Patterns.....	5
3.1.3 Analysis.....	6
3.1.4 Chosen Approach.....	7
3.1.5 Proven Feasibility.....	8
3.2 Web Browser.....	8
3.2.1 Desired Characteristics.....	8
3.2.2 Alternatives.....	9
3.2.3 Analysis.....	9
3.2.4 Chosen Approach.....	10
3.3.5 Proven Feasibility.....	10
3.3 Game Development Engine.....	11
3.3.1 Desired Characteristics.....	11
3.3.2 Alternatives.....	11
3.3.3 Analysis.....	12
3.3.4 Chosen Approach.....	13
3.3.5 Proven Feasibility.....	14
3.4 Programming Language.....	14
3.4.1 Desired Characteristics.....	14
3.4.2 Alternatives.....	14
3.4.3 Analysis.....	14
3.4.4 Chosen Approach.....	14
3.4.5 Proven Feasibility.....	15
<b>4.0 TECHNOLOGY INTEGRATION.....</b>	<b>16</b>
<b>5.0 CONCLUSION.....</b>	<b>17</b>
<b>6.0 REFERENCES.....</b>	<b>18</b>

# 1.0 INTRODUCTION

---

COVID-19 vaccination could play a critical role in not only improving the symptoms, but preventing hospitalizations and deaths related to such a disease as well, ultimately ending the pandemic situation. Yet, despite the possibility of such benefits, as of May 3, 2023, about 8.3 million U.S. children aged 12-17 and 17.4 million children aged 5-11 had yet to receive their first COVID-19 vaccine dose. In particular, over 60% of people under 20 years of age had yet to receive a single dose of the vaccine in Arizona. Now, even with the apparent need to promote vaccine uptake among adolescents, there are no current intervention studies underway aimed at improving vaccination rates among said youth.

And so, seeing such a gap within an egregious issue, with a background in video game development, Ashish Amresh aims to fill said gap with video games directed at the adolescent. In particular, we are designing a content agnostic game development framework that can be used to rapidly make games for different domains, such as COVID-19 and HPV. Now, in order to increase the likelihood of improving vaccination rates with such an approach, we are designing within the context of clinics where teens and their parents frequent and are waiting for their appointments. And within such a context with limited interaction time, the games must therefore be fast-paced, fun, and at the same time provide a range of decision making choices to fully engage the adolescents in addressing the outcomes. We will refer to the concept of fast-paced and engaging games as “burst games”; how our framework will focus on making games consisting of repetitive, quick “bursts” of gameplay that promotes learning by doing while minimizing the cost of failure and frustration for the player. Some examples of commercial burst games that we may follow are Angry Birds, and Puzzle Fighter.

Now, given that we are in the early stages of this project, we are currently in the process of analyzing key technological challenges, exploring possible alternatives, and choosing the most promising solutions out of said alternatives. Indeed, in this document, we begin by analyzing the major technological challenges presented by video game development, such as limitations within the game engine we choose to develop in. In the subsequent major sections, we will carefully analyze each of the alternative game engines, then converge and synthesize said analysis in a plan for integration.

## 2.0 TECHNOLOGICAL CHALLENGES

---

Now, to better foresee the major technological needs and challenges we will face, the constraints for the framework we will develop are as follows: the games must be short and repetitive in nature; the games will be played in a clinical setting via a tablet on the web browser; the games must result in a specific behavior change such as increasing vaccination rates; the games must target the adolescent with the possibility of parental supervision. Moreover, we will prototype two games, within the domains of COVID-19 and HPV respectively, using our developed framework as use cases for said framework. And so, with such constraints in mind, there are four immediate technological challenges that we must address.

- **Software Reusability:** Developing a framework that remains adaptable to various themes without being tied to specific content, ensuring flexibility and customization for different domains. The framework will have to be timeless, meaning that whoever has ownership will be able to reuse the framework for years to come.
- **Web Browser:** The burst games that are to be developed must be hosted on a tablet within a clinical setting i.e. while a patient is in the waiting room of a doctors office. The must be able to be played within a fifteen minute period and hosted on the web browser Itch.io
- **Game Development Engine:** The framework should be structured with easily identifiable components, corresponding to the game's objects, characters, effects, etc. It needs to be clear to anyone viewing the components how they interact with each other and what effects they have on connecting other components together.
- **Programming Language:** The framework must be developed using C# and the Unity game development engine according to client request. The source code for this framework is to be hosted on GitHub for ease of future development.

## 3.0 TECHNOLOGY ANALYSIS

---

Now we will explore the specific technological issues brought on by the needs and challenges of this project with the main technological challenges being: Software Reusability, Web Browser, Game Development Engine, and Programming Language. In particular, for each challenge, we will detail out desired characteristics, then investigate various alternative technologies for said challenge, using such desired characteristics as metrics for evaluating these alternatives. Next, after evaluating the alternatives for each challenge, one of these alternatives will be selected in addressing its corresponding technical challenge, then prove said alternative's feasibility by exploring how it will actually function in our framework.

### 3.1 Software Reusability

Software reusability is a fundamental concept in software engineering that emphasizes the development of modular components, libraries, and frameworks that can be utilized in multiple contexts, projects, or domains. By designing components that can be easily shared and leveraged across various games, the project aims to streamline the development process and minimize redundant efforts. To be specific, software reusability refers to a desired creational design pattern fit for our needs.

#### 3.1.1 Desired Characteristics

Given the nature of our proposed final product, being a framework to streamline the process of developing vaccine literacy games, it is integral that our desired characteristics correspond to intuitiveness and minimal effort. And so, we will strive to attain a total of three characteristics for our software reusability, totaling to a score of 15 points that an alternative can possibly reach. Such characteristics are as follows:

- **Supporting Maximum Reusability (5 points):** In order to develop a short and repetitive game with our framework, along with the prospect of more games in other domains being developed using this same framework, it is only natural that an ideal solution contains maximum software reuse.
- **Low maintenance (5 points):** Once again, to achieve maximum software reuse, an ideal solution should not be as involved when it comes to post-release patches. Ideally, one should be able to develop a game with our framework, then hardly have a need to look back and fix any glaring issues; how our framework should prioritize producing the least amount of issues as possible.

- **Easy to use (5 points):** Indeed, the whole point of “streamlining” a process is to make it as easy as possible. In our case, in order to streamline the development process of future vaccine literacy games, our framework must be intuitive to work with, and easy to couple desired components.

### 3.1.2 Alternative Design Patterns

- **Factory Method:** Also known as a Virtual Constructor. It works in such a way that it provides an interface for creating objects in a superclass, all the while allowing subclasses to generate their own types of objects. This method is mainly used when one does not know the exact types and dependencies of the objects, and when one wants to provide users of the framework with a way to extend its internal components.
- **Builder:** This design pattern lets one construct complex objects step by step. Moreover, this pattern allows one to produce different types and representations of an object using the same construction code; and it achieves such a feat through a Builder interface that declares construction steps common to all types of builders. The Builder pattern is mainly used to prevent a “telescoping constructor,” and when one wants to be able to create different representations of some product.
- **Singleton:** This pattern lets one ensure that a class has only one instance, while providing a global access point to said instance. All implementations of Singleton have the following two steps in common: making the default constructor private to prevent others from instantiating new instances with it; and designating a static method/function to act as a constructor, all the while calling upon the aforementioned private constructor. This pattern is mainly used when a class in one’s program should just have a single instance available to all clients.
- **Abstract Factory:** This design pattern lets one produce families of related objects without specifying their concrete classes. It earns its abstract title by declaring interfaces for a set of distinct but related products that make up a product family. The Abstract Factory is mainly applicable when one needs to work with various families of related products, but don’t want to depend on the concrete classes of said products.
- **Prototype:** This method lets one copy existing objects without making their code dependent on their classes. And it functions via a Prototype interface that declares cloning methods/functions; an interface which is delegated to the objects that must be cloned themselves. This design pattern is mainly used

when one's code should not depend on the concrete classes of objects that need to be copied, and when one wants to reduce the amount of subclasses that only differ in their initialization methods.

### 3.1.3 Analysis

Evaluation of these design patterns took the form of analyzing their corresponding applied examples, and seeing how such examples fared in accordance to our desired characteristics. Now, since all of the alternative design patterns are identified as creational patterns, it was expected that all scored relatively high, some more than others however, in the category of supporting maximum reusability. And because of such an expectation, the defining differences are in the categories of low maintenance and ease of applicability. See Figure 3.1.4 for a full breakdown of scores.

- **Factory Method**

- Supporting Maximum Reusability: Along with the ability to simply create new and slightly differing objects through other existing objects, it in turn saves system resources by reusing said existing objects, as they do not rebuild them each time.
- Low Maintenance: It utilizes the Open/Closed Principle; how new types of objects can get introduced without breaking existing infrastructure. However, due to the capability of introducing new subclasses with ease, the code may become more complicated and thereby harder to maintain.
- Easy to Use: It utilizes the Single Responsibility Principle; how one can move the product creation code into another place in the program, making the code easier to support. Moreover, it also avoids tight coupling between the creator and concrete products.

- **Builder**

- Supporting Maximum Reusability: It allows the reuse of the same constructor code when building the various representations of objects.
- Low Maintenance: There is still something left to be desired in this category, as the overall complexity increases since the pattern requires creating multiple new classes.
- Easy to Use: This one also utilizes the Single Responsibility Principle; how one can isolate the complex construction code from the business logic of the product. Moreover, the intuitiveness of this pattern is reinforced through its capability of constructing objects step-by-step.

- **Singleton**

- Supporting Maximum Reusability: The nature of this pattern limits reusability; there is still something left to be desired in this category.
- Low Maintenance: Compared to the other alternatives, this one scores the lowest in low maintenance, as it may prove difficult to test this pattern due to its lack of inheritance.
- Easy to Use: This design pattern violates the Single Responsibility Principle, as the designated Singleton class has to take on multiple responsibilities.

- **Abstract Factory**

- Supporting Maximum Reusability: Since the nature of this pattern is very similar to the Factory Method, how existing objects are reused to make more objects, it also scores relatively high in this category.
- Low Maintenance: Like the Factory Method, this one utilizes the Open/Closed Principle; how one can introduce new variants of objects without breaking existing code. But this one too suffers from the code becoming more complicated as new interfaces and classes are introduced.
- Easy to Use: Again, like the Factory Method, this pattern utilizes the Single Responsibility Principle; how one can extract the object creation code into one place, making it easier to support. Moreover, the nature of this pattern ensures that the objects produced by the factory are compatible with each other.

- **Prototype**

- Supporting Maximum Reusability: Objects can be cloned without coupling to their concrete classes. And the nature of this pattern allows one to get rid of repeated initialization in favor of cloning pre-built prototypes.
- Low Maintenance: The capability to completely discard repeated initialization code makes this design pattern stand out among the rest of the alternatives in this category.
- Easy to Use: Complex can be produced more conveniently via cloning. And such cloning serves as an alternative to inheritance when dealing with configuration presets for complex objects. However, cloning complex objects that have circular references may prove to be tricky to work with.

### 3.1.4 Chosen Approach



**Figure 3.1.4 - Desired Characteristics Scores of Alternative Design Patterns**

<b>Characteristic Scores</b>	<b>Factory Method</b>	<b>Builder</b>	<b>Singleton</b>	<b>Abstract Factory</b>	<b>Prototyping</b>
<b>Supporting Maximum Reusability</b>	5/5	5/5	3/5	5/5	5/5
<b>Low Maintenance</b>	4/5	3/5	1/5	4/5	5/5
<b>Easy to Use</b>	5/5	5/5	3/5	5/5	4/5
<b>Total</b>	<b>14/15</b>	<b>13/15</b>	<b>7/15</b>	<b>14/15</b>	<b>14/15</b>

Indeed, given that the Factory Method, Abstract Factory, and Prototyping are neck and neck in terms of overall score, choosing between these three design patterns proved to be arduous. And so, some elimination needed to be done. First, while the prospect of having the highest scores in the categories of reusability and low maintenance seemed promising, we ultimately value ease of use more given that our framework will be used by future developers; that is why we are eliminating Prototyping from the runnings. Now, we must choose between the two factory patterns; at this point, it is a matter of how involved our framework needs to be with other products. And so, **Team Medical Gaming Solutions has chosen the Factory Method as its designated design pattern**, as this pattern's applicability, such as not knowing the exact types and dependencies of our objects along with wanting to provide future developers a way to extend our framework's internal components, fits the exact bill of our project.

### 3.1.5 Proven Feasibility

Moving forward, we will experiment with various framework designs via UML diagrams throughout these initial phases of our project, all the while utilizing the Factor Method design pattern as guidance on good infrastructure. In particular, the two use cases, which are within the domains of COVID-19 and HPV respectively, that we will develop will demonstrate the effectiveness that the Factory Method has on our finalized framework.

## 3.2 Web Browser

With the task to make these games accessible within a clinical setting via tablet, the framework must be easily converted to an executable that can run on a web browser. It is the most efficient option for publishing games, as players can easily and quickly access games through a browser without the need to wait for a download.

### 3.2.1 Desired Characteristics

The desired browser must support our various game engine options, be easily and quickly accessible through the tablet in the clinical setting, and minimize the difficulty of playing the game within this setting. Each of our options must meet specific requirements when it comes to this challenge, measuring from a total of 15 points in each category.

- **Ease of Use (5 points):** The desired browser should make the game easy to access and play, as well as allow the developer to easily upload the games made with the framework. Since this framework is meant to be reused multiple times for different domains, the browser needs to have an easy to navigate UI, stable performance when uploading files, and simple customization for the game page.
- **Mobile Friendly (5 points):** Preferably, these games will be played on a tablet of any version. Therefore, the browser must allow the option for games to be played through a mobile device straight from the web page. We require support for IOS and Android, as well as some past version support to allow devices running on older versions to access the games.
- **HTML5 and WebGL Support (5 points):** These technologies are crucial for running modern web-based games and applications that include 3D graphics and interactivity. This will allow patients to have quick and easy access to a remote hosted game on the browser, and allow the developers to apply 3D graphics and assets within the games.

### 3.2.2 Alternatives

- **GameJolt:** A social community platform for game developers to share and publish their projects. Started up in 2002 by Yaprak and David DeCarmine as a way for independent creators to have a central platform to manage their content and communities.
- **Itch.io:** A website for users to host, download and share various forms of media, including video games and game related content. Launched in March of 2013 by Leaf Corcoran, and currently hosts over 700,000 products through their service.

- **Steam:** Founded in September of 2003, Steam is a digital video game distribution platform developed by Valve. It started as a storefront for Valve games specifically, until expanded into distributing third-party titles as well.

### 3.2.3 Analysis

We began researching these options through information pages included on the site, checking for any general information on the desired characteristics. Each browser has a dedicated FAQs page with top asked questions, as well as active community posts on specific software topics. Below are the results of each alternative in regards to how much of our set requirements they were able to satisfy. See figure 3.2.4 for a full breakdown of scores.

- **GameJolt**
  - Ease of Use: Relatively easy to use and upload files to.
  - Mobile Friendly: Allows specific games to be played on mobile devices, mostly on the mobile app version of the platform.
  - HTML5 and WebGL Support: Includes both HTML5 and WebGL support.
- **Itch.io**
  - Ease of Use: Relatively easy to use and upload files to, as well as customizability to help direct players on your page.
  - Mobile Friendly: Allows specific games to be played on mobile devices.
  - HTML5 and WebGL Support: Includes both HTML5 and WebGL support.
- **Steam**
  - Ease of Use: Slightly complex process of uploading games to the distribution platform.
  - Mobile Friendly: Includes third party mobile games, but cannot run them on the web page itself.
  - HTML5 and WebGL Support: No HTML5 or WebGL support, primarily hosts PC specific games.

### 3.2.4 Chosen Approach

**Figure 3.2.4 - Desired Characteristic Scores of Alternative Web Browsers**

Characteristic Scores	GameJolt	Itch.io	Steam
Ease of Use	5/5	5/5	3/5
Mobile Friendly	3/5	5/5	2/5
HTML5 and WebGL Support	5/5	5/5	0/5
Total	13/15	15/15	5/15

Overall, Itch.io scored the highest on each of our categories and appears to be our best option. GameJolt has more focus on mobile games within their mobile app, and less so on their web page. While it is still possible to play games straight from the web page with this browser, it is not as practical as playing them on a mobile device. Steam is a very popular choice when it comes to publishing games to distributors, but has little to no mobile support for this project. Itch.io has the most mobile friendly access to games, and is well known to be easy to use in terms of publishing content.

### 3.3.5 Proven Feasibility

With the results shown above, **our best chosen approach is Itch.io to publish and upload our games onto.** It includes the necessary technologies, being HTML5 and WebGL, and also supports mobile device gameplay from the web browser. Itch.io also has an easy process of uploading games and related content to their platform and allows creators to customize their pages, which will be helpful when directing players to the desired information on the game.

## 3.3 Game Development Engine

To develop these burst games, a game development engine is needed to implement necessary requirements. A development engine will provide a GUI for the user to interact with the game as well. Two use cases (video games) will be developed over the course of this project which will require the analysis of the most optimal game engine.

### 3.3.1 Desired Characteristics

One of our main deliverables will be developing two burst games to serve as use-cases for the architectural design framework. Burst games are fast-paced, repetitive games that have a minuscule amount of frustration involved. Examples of these games would be *Angry Birds* or *Puzzle Fighter*, which is a Tetris-style game where you compete against another. To allow for reusability and ease of development for future games, three

characteristics are desired for an engine, measuring up to 15 points when deciding which one will best fit our requirements.

- **Component oriented ‘ECS’ (5 points):** The engine must be component oriented to allow other owners to implement their own healthcare games with ease i.e. choosing the component according to their demands
- **Extend the base components of the engine (5 points):** The components should extend what is already in the game development engine in order to save effort and time during development. It is not necessary to create completely new ones (client request).
- **Web browser support (5 points):** The engine should be able to export the use cases with web browser compatibility in order to host the game in a clinical setting via web browser. Other build support options would also be beneficial to allow for flexibility with hosting in the future i.e. PC application, mobile application.

### 3.3.2 Alternatives

- **Godot:** A popular, free and open-source game engine under the MIT license. It was originally developed by Argentine developers Juan Linietsky and Ariel Manzur for companies in Latin America prior to its public release in 2014. Now, Godot is mainly used by indie developers for its lightweighted-ness. It has a simple interface for beginner developers and provides its own IDE. Games such as *Cassette Beasts* (2023) and *Deponia* (2016) have been made using Godot.
- **Unreal:** One of the most well-known 3D game engines developed by Epic Games. It was first showcased in 1998 via the first-person shooter game “Unreal.” Now, Unreal is mainly used by high-profile video game studios and AAA developers; moreover, Unreal is also used in film making with its robust rendering and lighting systems. Unreal has played a development role in such games as *Kingdom Hearts 3* (2019) and *Street Fighter V* (2016).
- **Unity:** Another well-known game engine; it was developed by Unity Technologies, and first announced and released in June 2005 as a Mac OS X game engine. The engine has since been extended for cross-platform support. Now, Unity is predominantly used by indie developers for its versatility and

support for a variety of desktop, console, mobile, and virtual reality platforms. Two games that were developed in Unity are *Among Us* (2018) and *I Am Bread* (2015).

### 3.3.3 Analysis

Some essential elements of an engine needed for this technological challenge is for the engine to be component based, have a sufficient asset library, and be able to have web browser compatibility. Points were reduced according to the requirements that the engine possesses or lacks. Each engine listed below will outline its advantages and disadvantages.

- **Godot**

- Component oriented: Godot does not use an Entity Component System and instead uses ‘nodes’. Godot chooses to embrace OOP and these nodes contain both data and logic. Nodes allow for reusability, modularity, and flexibility during development. Godot does composition at a higher level than in a traditional ECS.
- Extend the base components of the engine: The Asset Library, or AssetLib, in Godot is a repository of user-submitted resources for developers.
- Web browser support: Godot supports exporting developed games to HTML5 platforms.

- **Unreal**

- Component oriented: Unlike other engines, Unreal uses an actor component system. This system requires game objects to be represented as ‘actors’ that have components attached to them. These actors carry out the role of containers for components
- Extend the base components of the engine: UActorComponent is the base class for all components in the Unreal engine. Unreal 4 will allow a developer to inherit from game base class i.e. PlayerController, GameMode, HUD, etc.
- Web browser support: HTML5 has been depreciated since the 4.23 version of Unreal

- **Unity**

- Component oriented: ECS is the core of Unity. It constructs complex game objects by attaching and combining reusable components. It also provides memory control and eliminates refactoring that would have been necessary with object-oriented architectures.

- Extend the base components of the engine: It is possible to extend existing components from Unity's base components. An example of this would be adding on a onHold event to a basic Unity button.
- Web browser support: Includes features for building both HTML5 and WebGL applications.

### 3.3.4 Chosen Approach

Characteristic Scores	Godot	Unreal	Unity
Component Oriented	3/5	3/5	5/5
Extend Base	4/5	5/5	5/5
Browser Support	5/5	0/5	5/5
Total	12/15	8/15	15/15

Game engines have very similar features that are attractive to small development teams such as simple interfaces, free licenses, lightweight-ness, etc. Godot received a 12 out of 15 due to the lack of an ECS and the lesser component library compared to Unity's. Although the Unreal engine produces many AAA video games, their extensive features are not necessary for our project. Unreal also does not support web browser gaming in their newer versions. Ultimately, we have decided to select Unity as our development engine based on client request and overall desired characteristics. Unity will provide the best services for our project requirements such as having an entity component system, the ability to extend the engines base components, and supporting a web browser as a host for a developed game.

### 3.3.5 Proven Feasibility

**With Unity being our desired game engine**, one major technological issue is figuring out how to extend Unity's entity-component system to develop the various content agnostic components of our framework. In particular, some functions we see needing in order to resolve such an issue are as follows: a public placeholder class representing a component; various options for a component to interact with the environment; and a way of combining and coupling these components. Now, in terms of the final product, these functions, along with the others to be designed in the future, should play the role of providing a streamlined process of developing any vaccine-literacy burst game within our framework. We will utilize Unity as a game development engine by producing a tech demo by the end of the Fall 2023 semester, developing two use cases by the end of the project, and the framework as well.

### 3.4 Programming Language

In order to develop a game using the Unity engine, the programming languages are limited to its native C#, JavaScript, and Boo.

#### 3.4.1 Desired Characteristics

- The programming language we choose should integrate with the Unity environment, enabling us to handle intricate game mechanics effectively. Additionally, it should offer good debugging tools and possess an active and supportive online community for troubleshooting purposes.

#### 3.4.2 Alternatives

- The top options for the main coding language used on our project were C#, JavaScript, and Boo. While each has its pros and cons, C# stands out as the more popular choice, especially given its strong compatibility with Unity's framework and its extensive documentation and support resources.

#### 3.4.3 Analysis

- Through research we found that C# is liked within the Unity development community mainly because of its versatility and extensive amount of features when working with game dev languages based in C. On the other hand, JavaScript is good but, lacks some of the robust capabilities offered by C#. Boo is another good option, but doesn't really enjoy the same level of popularity and community support as C#.

Characteristic Scores	C#	JavaScript	Boo
Low Maintenance	4/5	3/5	1/5
Easy to Use	5/5	5/5	3/5
Total	9/10	8/10	4/10



### 3.4.4 Chosen Approach

- Considering its the wide range of usage, the compatibilites with Unity, and of course the wealth of online resources, we chose to use C# as our primary programming language. This decided this to ensure a seamless game development process with proficnet support and guidance available when needed.

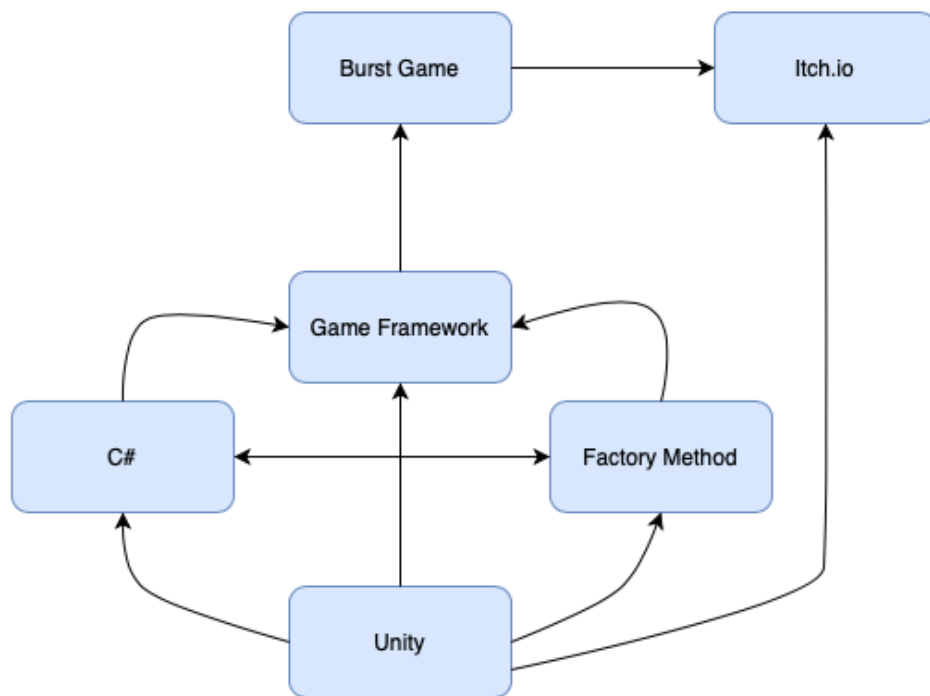
### 3.4.5 Proven Feasibility

- Through thorough testing and experimentation within our Unity environment locally, we confirmed that C# effectively meets the needs for our game development requirements. Our extensive research and practical exploration have reinforced our confidence in C# as the most suited and practical choice for our project.

## 4.0 TECHNOLOGY INTEGRATION

---

With our chosen programming language, web browser, design pattern, and game engine to establish this project upon, we can plan out how we will integrate our burst game framework using these tools. The main goal will be to create a reusable framework that developers can use to make burst games on their desired domain, so the framework must be flexible enough for anyone to effortlessly modify the components of the game. Any text-based and graphics-based content within the games must be modifiable as well, to keep it from being restricted to a limited number of domains like HPV and COVID-19.



**Figure 4.1 Integration Diagram**

The diagram above shows the connections between each of our tools and how the process of making the burst games will work. The central module is our game framework, where our main tools will assist us in developing. Our chosen game engine, Unity, will be the base on which all of our development takes place. C# is the main programming language we will be using to develop the components, so it must be connected to the Unity

module for us to be able to develop within the game engine. Our developers on the team will handle the backend development of the framework to handle the gameplay mechanics and how they connect to the other components of the framework. The development within Unity must also follow and connect to the factory method design pattern, which will make our structure as efficient as possible when programming. We also need to connect the C# module to our factory method module. This way, all code written for the components will remain consistent for development and can be easily followed when reading code. With a framework completed, there will need to be two test games developed to show the effectiveness of the framework itself. Everything within the game framework connects to the burst game module itself, since the games must be developed with the components we make within the framework. Our last module, Itch.io, is the browser we will upload the games onto, allowing players to access the game straight from their device. Unity connects to the module as well, to ensure we have WebGL and/or HTML5 support in the engine to satisfy the requirement of making these games playable on the Itch.io browser itself.

The two test games should follow the burst-style format of being fast paced, simple, easy to play and understand, and minimizes frustration for players on restarts. The games can either focus more on puzzle or action gameplay, as long as they follow those aspects of the burst format. The framework must be developed in a way that allows us to switch out components, layouts, text, and any other content in the game that will let us develop two different types of burst games with ease. Components need to be developed modularly to keep the overall framework intact if any are to be removed and/or modified. The factory method design pattern will also assist in making the components easy to implement, test, change and reuse.

This integration is how we plan to effectively create our framework for burst style games to be developed in, with the goal of raising COVID-19 vaccination rates among teens and children in the U.S.

## 5.0 CONCLUSION

---

Indeed, it is unfortunate that millions of adolescents have yet to take their COVID-19 vaccines; and it is equally as unfortunate that there are currently no intervention studies and efforts to increase the vaccine uptake among the youth. That is why we, team Medical Gaming Solutions, will help Dr. Amresh in performing the first ever intervention study aimed at improving such vaccination rates for the adolescent. All of this through a video game framework that is low maintenance, utilizes maximum software reuse, and intuitive to use, such that the framework streamlines the process of developing vaccine literacy games, or any medical literacy games for that matter, in the future.

But of course, no issue with a proposed solution is complete without a plan for development. For starters, we will develop our video game framework in Unity. Given Unity's versatility and extensive cross-platform support, along with the fact that all of our team members possess fundamental skills in developing within Unity, it becomes apparent that Unity is the obvious choice for a game engine to develop in. Now, while Godot seems like a promising alternative, the tradeoff is that an extensive amount of time would be needed for all team members to learn Godot; such time, of which, we cannot risk losing in this tight two-semester schedule. Now, as for finding a way to host our games in a clinical tablet setting, we plan to simply host them in a web browser with our preferred hosting page being Itch.io.

And so, with such a plan going forward, the two use cases we will develop using our framework, within the domains of COVID-19 and HPV respectively, will demonstrate how our framework is used with minimal effort and maximum software reuse.

We hope that this first, but very crucial, step in Dr. Amresh's vision will result in those millions of adolescents finally keeping up with their vaccines, and ultimately end this pandemic situation.

## 6.0 REFERENCES

---

Shvets, Alexander. *Dive Into Design Patterns*. Refactoring Guru, December 5, 2018.