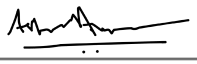
MEDICAL GAMING SOLUTIONS

LEVELING UP HEALTHCARE THROUGH GAMING

Sponsor 

Team Lead 

Requirements Specification

Version 1.2

10-24-23

Project Sponsor: Dr. Ashish Amresh

Faculty Member: Italo Santos

Team Members:

Rain Bigsby, Veronica Cardenas,

Ethan Ikhifa, Lenin Valdivia

This team document will describe our projects functional, non-functional, and environmental requirements to be fulfilled during development. It also analyzes our solution, potential risks, and project timeline for our framework.

Table of Contents

1.0 INTRODUCTION **2**
2.0 PROBLEM STATEMENT **3**
3.0 SOLUTION VISION **4**
4.0 PROJECT REQUIREMENTS **5**
4.1 Functional Requirements 5
4.1.1 Knowledge Drop Components 5
4.1.2 Game Mechanics Repository 7
4.1.3 Data Management 9
4.2 Performance Requirements 12
4.2.1 Usability 12
4.2.2 Optimization 12
4.2.3 Compatibility 13
4.2.4 Maintainability 13
4.2.5 Scalability 14
4.3 Environmental Requirements 15
4.3.1 Unity 15
4.3.2 Clinical Tablet Setting 15
5.0 POTENTIAL RISKS **16**
5.1 Our System is Computing Intensive 16
5.1.1 Cause of Risk 16
5.1.2 Likelihood 16
5.1.3 Future Steps 16
5.2 Undocumented Dependencies 17
5.2.1 Cause of Risk 17
5.2.2 Likelihood 17
5.2.3 Future Steps 17
6.0 PROJECT PLAN **18**
7.0 CONCLUSION **19**
8.0 GLOSSARY **20**

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 [1]. Over 60% of people under 20 years of age had yet to receive a single dose of the vaccine in Arizona [2]. On top of such COVID-19 vaccination rates, HPV vaccination continues to be in a dire situation; as the Center for Disease Control and Prevention reports, approximately 80 million Americans are currently infected by human papillomavirus (HPV). Now, even with the clear need to promote vaccine uptake among adolescents, there are no current intervention studies underway aimed at improving vaccination rates among said youth.

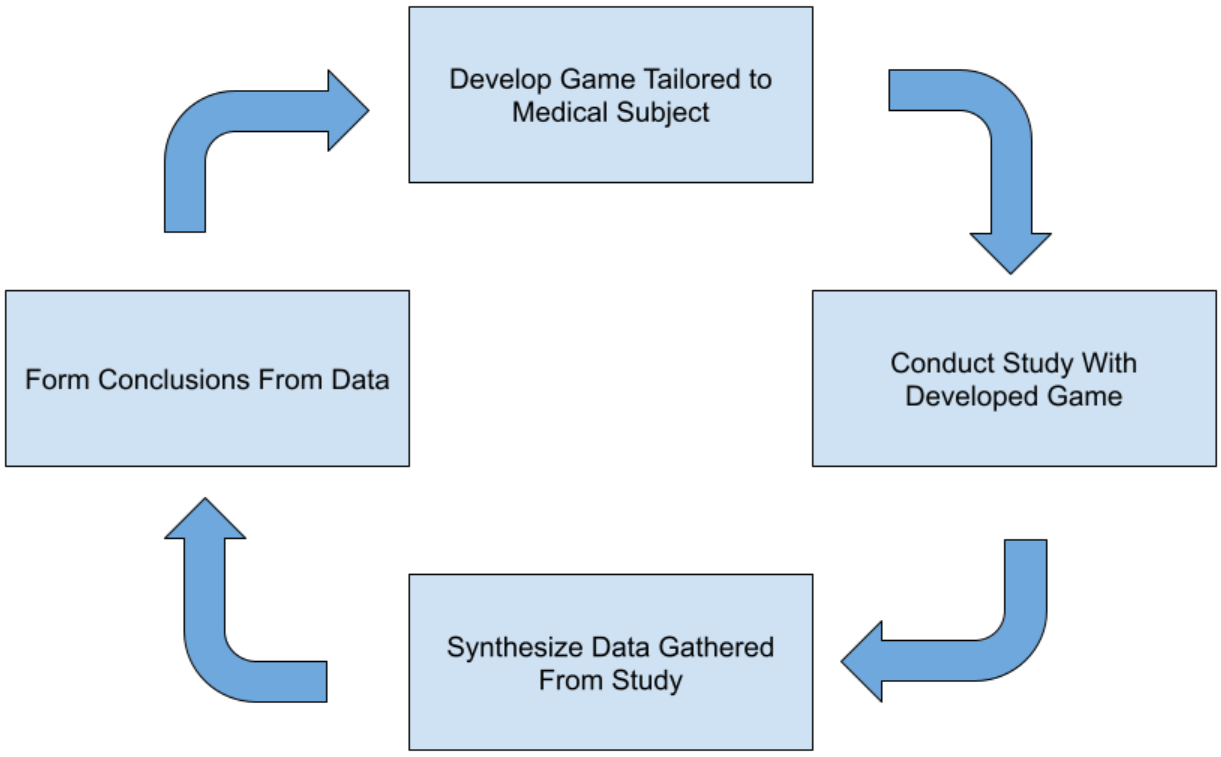
That is where our client, Ashish Amresh, comes in. Indeed, being an associate professor at the School of Informatics, Computing, and Cyber Systems (SICCS), Dr. Amresh is currently conducting research in the development of software tools to improve game-based learning outcomes. In the case of vaccination rates among the youth, Dr. Amresh aims at improving said vaccination rates via an intervention study, and quite possibly the first, targeting the overall vaccine literacy of the adolescent through video games. Now, to increase the likelihood of improving vaccination rates with such an approach, Dr. Amresh plans to conduct said study within the context of clinics where teens and their parents frequent and are waiting for their appointments. Within such a context with limited interaction time in mind, the video games for this study 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 the games developed for this study will consist 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, in this document, we will go over the glaring issues that correlate to the video game development aspect of this project, then propose a solution vision to said issues. In the subsequent major sections, we will go into further detail about the functional, performance, and environmental requirements of our proposed solution, lay out relevant potential risks of said solution, then propose project milestones we intend to achieve going forward.

2.0 PROBLEM STATEMENT

Given that the core of our client's project is to smoothly integrate video games and improved vaccination rates together through an intervention study, the following graphical representation [Figure 2.0] of our client's proposed workflow gives insight into how our client plans to conduct his project "smoothly."

Figure 2.0 - Client's Workflow



The bottleneck of such a workflow is the development of the video game. Indeed, the issues imposed by developing a video game tailored to a medical subject from scratch are as follows:

- Substantial amount of time is taken up.
- Unnecessary resources, such as money, are spent.
- The studies our client wants to conduct with these video games may be bogged down by slow video game development.

3.0 SOLUTION VISION

To decrease the amount of development time for the video game to not impede upon the client's workflow, our client needs a way to streamline the development process of such video games. And so, our proposed solution is to develop a content agnostic video game framework such that any developer may use it to expedite development of the video game by building on top of the framework, and filling the content agnostic components with whatever they need. The following is a list of specific features pertaining to our proposed framework.

- Blank/placeholder knowledge-drop components.
- Game structure templates.
- Extendable classes/objects.

We will develop our framework within Unity; ergo, future developers using our framework will have to work within Unity as well. Now, since our client will conduct research using the video games developed with our framework, it must provide a method that allows developers and researchers to easily create and keep track of their own data parameters. It is important to keep in mind, however, that nothing needs to be concrete; our framework should be as open-ended as possible so as to not inhibit the design-freedom of future developers. Therefore, we will only provide skeletal structures of these data components within our framework. Now, as for the change to our client's workflow when our framework is integrated, less time and resources will be spent on video development via future developers building on top of the provided game mechanic components within our framework. Indeed, less time and resources spent on video game development will result in rapid deployment of such video games, ultimately allowing our client to conduct his studies as soon as possible. See Figure 3.0 below for a graphical representation of such a workflow. One tradeoff, however, is the minimal creative freedom our framework will provide future developers, as these developers will only be extending and building on top of our framework to develop their video games.

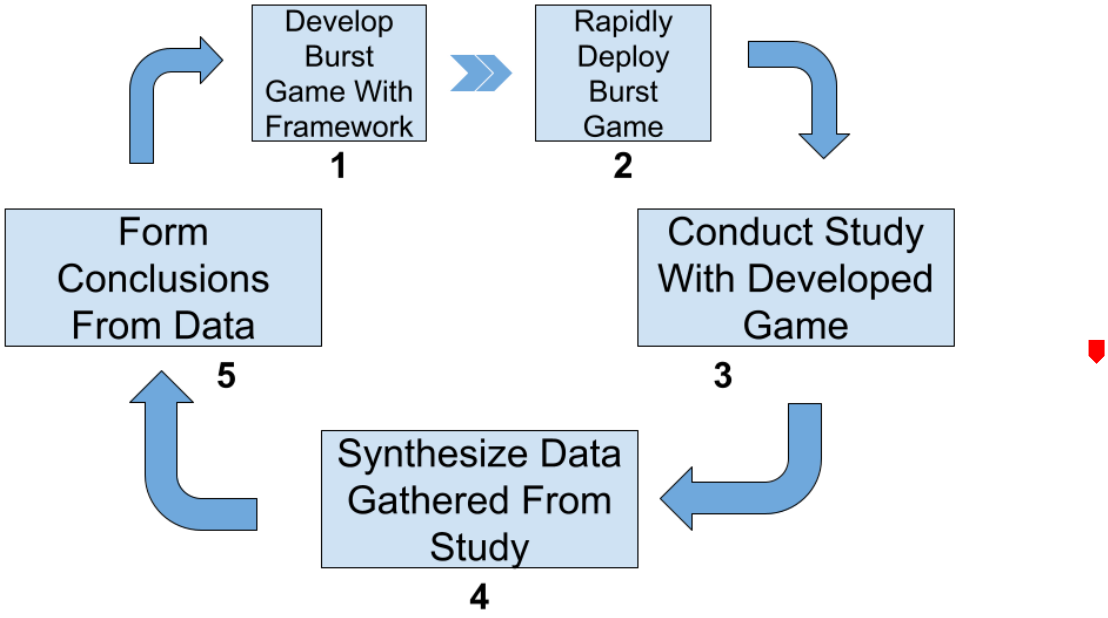


Figure 3.0 - Improved Client Workflow

4.0 PROJECT REQUIREMENTS

To provide a framework for future developers that will enable them to develop video games with minimal effort and maximum software reusability, we must outline certain requirements for this project. Such project requirements that are necessary for satisfying our goal of seeing an increase of vaccination rates in adolescents will be divided into the following three categories: functional requirements, non-functional requirements, and environmental requirements. Functional requirements will discuss what our framework will accomplish, non-functional requirements will examine how well the system will perform, and environmental requirements will review what software, hardware, and design strategies that are imposed by the sponsor.

4.1 Functional Requirements

Functional requirements indicate what team Medical Gaming Solutions assures the framework will provide to future developers. The following subsections are what we believe will make up the core of capabilities our framework will possess.

4.1.1 Knowledge Drop Components

To incite a behavior change outcome for the target player base, such as increasing vaccination rates among younger adults, our framework must allow developers to easily incorporate their desired medical topic they wish to educate the player on into their video game. In this section, we will review the processes encompassed by the implementation of these knowledge drops throughout the game; how such processes consist of knowledge drop templates, and managing custom created knowledge drop components**.**

- Knowledge Drop Templates

By providing knowledge drop templates in which developers can straightforwardly add the template to their game and fill it with the content they need, the process of integrating a topic to the video game is further expedited. The following will go over the functionalities of such knowledge drop templates.

- **Provided templates:** The knowledge drop templates in which our framework will accommodate developers are as follows.
    - The “game over knowledge drop”: the idea behind this one is that when a player fails a specific task or a level in the game and sees the gamer over screen, such a screen will also have a content area in which the developers can fill in with whatever fact or statistic relating to the topic of the game.

    - The “success knowledge drop”: the principle behind this one is the same as the “game over knowledge drop,” except that the content area will be provided upon the player succeeding on a specific task or level.
    - Finally, the “knowledge dropping Non-Playable-Character (NPC)”: this template is simply an NPC with a text box as a content area, such that developers can insert said NPC at any point in the game.

- **Add template to game world/scene:** Through a combination of Unity gameObjects and prefabs of the provided templates, developers will be able to easily append such knowledge drop components throughout their game as needed.

- **Fill in content areas:** Once a knowledge drop template has been implemented within the game scene, the content areas of such templates will appear with “lorem ipsum” by default. Then, developers will be able to simply edit the content areas via Unity’s UI system. Indeed, this will be possible given that the content areas of the templates will be implemented as Text UI elements.

- Managing Custom Knowledge Drop Components

Along with knowledge drop templates to choose from, our framework will also provide a way for developers to extend and create their own knowledge drop components. Indeed, while the nature of developing on top of a framework limits a significant amount of creativity, extendable knowledge drop components will lessen the creative limit on how future developers want to present their desired topic within their video game.

- **Blank prefab component:** While having a blank prefab may seem as an inefficient use of resources, the idea behind providing such a prefab is to suggest to future developers that they can craft their own knowledge drop components with that blank prefab. Now, whether they find the provided templates inadequate in presenting their desired topic, or they merely want to enforce their creativity on the game, developers can customize and add to these blank prefabs through Unity’s GameObject system, and the C# script attached to said prefabs.

- **Add custom knowledge drop to scene:** The custom knowledge drop components may be added to the game scene exactly as how one would add one of the template components. The crux of this particular functionality is that the developer will only need to edit one instance of the custom knowledge drop component to configure all instances of it throughout the game simultaneously.

4.1.2 Game Mechanics Repository

To successfully present a framework for developers to modify according to their preferences, a game mechanics repository will be required in our framework design document. Game mechanics are a set of rules that guide the player's actions while in game and the response that the game returns in consequence of the player's action. To differentiate between game mechanics and game play, we will refer to game play as the set of core game mechanics that make the game a whole entity for a player to interact with.

- Fast Paced and Engaging

In the video game *Super Puzzle Fighter II Turbo (1996)*, the player is required to quickly figure out where to place a random block before the next one appears, taking up space.

Super Puzzle Fighter II Turbo (1996) [3]

The player is put up against an opponent, creating a sense of urgency and an objective for the player. This burst game contains such game mechanics as a winning condition, quick time events, score points, and controller movements for the puzzle pieces.

In our case, our framework will use an "endless runner" prototype as its use case during development. An example of an endless runner video game would be *Subway Surfers (2012)* or *Jetpack Joyride (2011)*. This style of game allows the player to infinitely run across an environment collecting items and dodging obstacles. Scoring is implemented in popular endless runner video games that factor in how long a player has been alive in game.

Jetpack Joyride (2011) [4]

Game mechanics that could be included in our 2D endless runner prototype that will make the video game quick-paced and engaging would be:

- **Movement**: Players will be able to navigate through the environment via tapping or swiping on the screen.

- **Obstacle Variety**: Differing obstacles will be present during the game's runtime such as platforms to jump on or duck under depending on the player's decision. Obstacle implementation will depend on the developer's discretion.

- **Powerups**: Items will be available for the player to obtain during their course of playing. These powerups will differ for the developer depending on their use case. For example, one could implement a speed boost, regain health, or obtain currency.

- **Single-Objective Challenges**: Simple and clear objectives can motivate the player to engage with the gameplay and feel a sense of accomplishment when completed.

- Short and Repetitive

Burst games are also characterized by it's repetitiveness and short play time. Since our framework requires the prototype to be in a clinical setting with fifteen minutes or less of play time, some game mechanics are preferred over ones that require complex and long playthrough strategies. Repetitive gameplay allows for accessibility, meaning that any player of any skill level can participate.

- **Player Controls**: To allow for differing skill levels across players, simplifying controls to a one-tap system will mitigate the learning curve of the navigation game mechanic. Quick actions like swiping

and tapping are preferred for games with a short play time. It also provides repetitive motions for the player to execute.

- **Progressive Difficulty**: Levels can get increasingly more challenging as the game progresses so that a player can feel challenged as they play the repetitive game.

- **Respawn**: If a player loses, they can continue playing by respawning. This reduces the downtime between attempts which is crucial for the limited time they have to play in the clinical setting.

These are rough suggestions on what our gameplay would include for our prototype based on our framework. Detailed game mechanics will be provided on the game design document delivered to the client. Developers will be able to pick and choose which components align with their selected genre. Game mechanics will be implemented via components in Unity.

4.1.3 Data Management

Components within the framework will also need to hold or possibly transfer various types of data related to the game. Any data within the game should be capable of being effectively accessed, collected, and organized to ensure the best performance for the player. The data management in the framework itself will be less customizable to the developers, and should instead be pre-developed and be available for utilization in the components of the framework.

- Score Data

One of the main incentives for patients waiting in the clinical areas to want to try the game will be the scoring system. Scores give players not only a goal to work toward, but also a sense of satisfaction when continuously reaching larger and larger numbers as they play. The data for this score system should be transferable between certain components of the framework to accurately display scores to the user depending on which menu they are viewing.

- **Player Ongoing Score:** The player should be given a visible score for their current run somewhere on the screen that does not take up too much space, avoiding distraction for the players. In order to properly manage the current player's score, there should be factors from the player's character (run time, enemies hit, power ups collected, etc.) calculated for the final score displayed at the end.

- **Leaderboard Management:** The leaderboard is the component that should contain all the previous scores of players and display the top scores to the player either at the end of their playthrough, or, if a leaderboard menu button is included, should be displayed when selected by the user. This component will contain any related data to the specific player's score, such as a name or initials. Clinical settings are not guaranteed to always have a connection to a specific database for this leaderboard, so the data should be stored locally, and only include the players scores on that device.

- Character Data

The playable character for the users will also require some data to be accessed by different components of the game. Since the player will be interacting with almost everything in the game, most of the objects will need to be able to access the character data to activate appropriate responses when the player interacts or collides with them.

- **PowerUp:** Various power ups will be available for the player to collect as they play through the game. These power ups and their given abilities may be stored in a separate component, but the player character should access the specific data for which power up they currently possess. This data should also be accessible by outside components, such as enemies or obstacles, so they may behave according to the player's power up interactions.

- **Current Position:** A crucial piece of data contained in the character data is the current position. This must be known by everything within the game to keep an immersive world for the player to interact with. Obstacles and platforms will use this data for collisions from the player, or giving them surfaces to walk on without clipping through them. Enemies will use the position as a hitbox to know when they are in range of the player and can appropriately send them to the game over screen if they lose. The figure below [Figure 4.1.3 A] depicts how jump and slide movements should change the position data for obstacles and platforms to not collide with the player, and recognize that the character can avoid it, or possibly land on it.

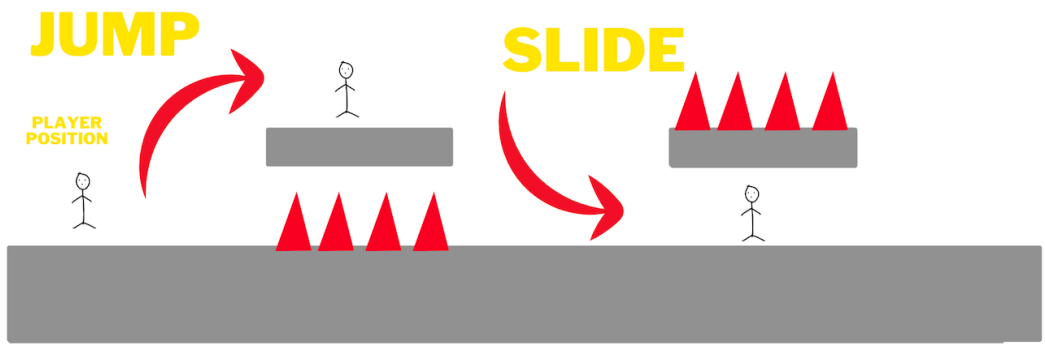


Figure 4.1.3 A

- Research Data

The main goal for this project will require some researchers to analyze data from the players during their playthroughs. This will give them a better understanding of the decision making outcomes for teens and children in a game based environment, and how they react to it.

- **Consenting Form Component:** An option to better organize the game data relating to specific players and give clinics value in allowing these games in their waiting rooms, easing the consent form process. Developers will be able to insert their custom form into the component for the players to fill out, and can then be printed as a PDF for the clinic to use. Data given by the player can be used in game to identify them with a unique ID based on their name, without displaying their real name or any sensitive information.

- **Research Component:** The main component that will be used for researchers to analyze player actions. Will be customizable to detect certain interactions (opened a specific door, destroyed a type of enemy, etc.) and count the total times it happened on a run. Researchers can then use this component to look at a specific player's ID and analyze their specific actions.

4.2 Performance Requirements

Now that we have covered what exactly our system is expected to do via functional requirements, we will now go over how such functional requirements are expected to perform within certain criteria; we will refer to such criteria as performance, also known as non-functional, requirements. And so, to ensure that our framework is satisfactory, we will prioritize the following performance requirements: **usability, optimization, compatibility, maintainability, and scalability.**

4.2.1 Usability

The product is expected to have little to no issues when it comes to the usability of the framework. We must account for both developers using the framework for video game development, as well as the patients who will be accessing and playing the games made with our framework.

- Developer Use: The framework should have minimal difficulty for developers to customize the content within the components. Ease of use should be assisted with simplicity in the framework, component descriptions on how they should function, and descriptive comments to guide the users how to modify the content. For example, the customizable knowledge drops should provide clear comments on which parts of the component can be swapped out for the developers chosen content, without affecting the component itself.

- Clinical Setting: Coinciding with the goal to increase vaccination rates among teens, these games must be playable within a clinical setting for patients in the waiting rooms. Tablets located within these rooms will be able to access these games via a web browser (Itch.io) and can play the game locally within that browser. This will reduce any waiting times for downloads or installing the application when the players will already have a limited time to try the game.

- Controls Accessibility: Unlike home consoles or gaming specific personal computers, the controls will be quite limited through a touch screen tablet. This requires us to make sure we get the most out of the few control inputs we can take in for the player. We can ease the difficulty in using these controls for them by keeping controls restricted to swiping up, down, left, and right without diagonal controls. Therefore, if the player swipes slightly to the right when swiping up, it will not read it as a completely different function, like an ‘up-right’ input, and cause frustration for the player.

4.2.2 Optimization

The performance of the framework needs to be at an optimal level as to run at a clean fast rate, but not overwhelm the device the users will use to play the games.

- Simple Shapes: We must keep in mind that these games will be played on simple tablet devices and should not have to account for high definition graphics or high object counts. Therefore, maintaining the framework as non-demanding as possible with simple shapes for objects is the best suited way to keep users engaged without running into performance issues.

- Static Objects: Another way to keep optimization at its best performance is to keep objects that only need visual representation (trees, NPC's, etc.) as static objects. This will reduce the unnecessary performance issues that could occur from too many active objects at once.

- Assets: The framework would benefit most by using assets that are already compatible with the framework and are known to work efficiently within the game engine environment. Luckily, our chosen game engine, Unity, includes the Unity Asset Store for us to utilize within our framework.

4.2.3 Compatibility

Developers will most likely not find interest in a framework that is not compatible with some of the most famously used software. Therefore, we shall ensure that the compatibility of our project satisfies enough products to give developers an incentive to experiment with the framework.

- Browser: The framework should be compatible with major web browsers, such as Chrome, Firefox, Safari, and Edge, to ensure a consistent experience across different platforms.

- Devices: Our framework should be optimized to run smoothly on clinical tablets, the designated devices for game deployment. Compatibility with different tablet models and specifications is essential for widespread usability. The framework should build games compatible with at least some of the most popularly used versions of tablet devices available today.

4.2.4 Maintainability

Maintainability highlights how the framework can be updated, modified, and sustained over time. A maintainable system reduces the burden on developers and ensures the longevity of the framework.

- Clean Coding Standard: One of the major steps in maintaining the framework is ensuring the code is easily readable with our chosen software architecture. We should avoid complex logic and long nested statements as much as possible, as well as include human readable names for important variables.

- Documentation: The documentation we will conduct on our project will also assist us on maintainability. They can work as a frame of reference on what we set to accomplish at the time, what our plans for certain segments were, or any other necessary information we include. Documentation also helps us keep track of any previous plans we have written that need to be updated or removed from the document if any changes occurred.

- Version Control: Keeping our past versions of files in a repository is the safest way of maintaining our code and avoiding any issues that occur during development. Using a version control system (most likely GitHub), we can keep track of any bugs halting development, and where they occurred in our version history. This also allows us to safely implement new features when we desire to, without overwriting the previous versions, and avoid losing any version data that should be kept.

4.2.5 Scalability

The scalability of our framework is very important to accommodate potential growth in both the number of games and our current amount of users. Designing the framework with scalability in mind makes sure that it will be able to handle the increasing demand without sacrificing performance. This is what is needed for a scalable yet effective framework for maintaining a robust project as it evolves.

4.3 Environmental Requirements

Having gone over the exact capabilities of our system, and how such capabilities will perform within our designated criteria, we must now explore the constraints encompassing our project; in other words, we will now go over environmental requirements. And so, the following subsections describe both the software and hardware restrictions that the team must consider throughout the development of our framework.

4.3.1 Unity

We will have to develop our framework within Unity; this is a constraint imposed by our client, as Unity will also be his choice game engine for the burst games to be developed for his studies. Indeed, while other game engines, such as Godot, may seem desirable, we must align with our client's vision, all in order to ensure that the video game development process for future developers is successfully integrated and streamlined into our client's workflow [see Figure 3.0 for this workflow].

4.3.2 Clinical Tablet Setting

Along with the software restriction of having to develop our framework within Unity, we must also keep in mind how the video games, which are going to be developed with our framework of course, are going to be played; how our client plans to conduct his studies within a clinical setting. In particular, it is expected that the clinics will provide the target players with tablets to play the video games on. And so, while designing our framework, we must consider the hardware limitations of a tablet. Therefore, our system cannot be too resource demanding, otherwise the video game interactions needed from the target players for our client's studies will be compromised. See the following section for an in-depth view of the risks involved with developing within the clinical tablet setting.

5.0 POTENTIAL RISKS

In any project that spans a significant amount of time, an entire school year in our case, there are bound to be risks to consider. And most of the time, such risks are a result of human error; how our disposition to make mistakes as humans may cause our client a significant amount of time and resources lost while conducting his research. Indeed, we will explore the most probable and egregious risks in this section.

5.1 Our System is Computing Intensive

All of the games to be developed with our framework will presumably only run on a clinical tablet provided to the target player during a small interaction window of about 15 minutes. If we develop our system in such a way that it bloats the amount of computing resources required to run a game, then the amount of actual playtime the target player will engage in will be limited due to the insubstantial computing power of a tablet. And in the worst case, the target player will not be able to play the video game at all.

5.1.1 Cause of Risk

Negligence: As the development process of the framework moves along, frequent resource-usage benchmarks should be conducted accordingly. Failure to upkeep on such benchmarks may result in bloated and unoptimized software.

5.1.2 Likelihood

Moderate: It can become very easy to neglect the resource-usage benchmarks after not conducting such a benchmark once when we were supposed to. Indeed, this will become a matter of consistency and attention from the team throughout development.

5.1.3 Future Steps

Regardless of whether the amount of playtime is limited or it's the worst case scenario of zero playtime, we will have to go back to the drawing board on the framework and painstakingly rework the system; all the while, causing a significant amount of time and energy wasted for both the team and the client. We wish to avoid this by any means. In response to the identified risk associated with computing intensity and potential limitations on playtime, our team has devised a comprehensive mitigation plan to address these concerns and ensure the smooth development of the framework. The proactive approach outlined below aims to avoid the worst-case scenario of limited playtime.

5.2 Undocumented Dependencies

To strive for maximum ease of use, our framework must be as intuitive as possible. Such intuitiveness may be achieved through extensive documentation on core features of our framework, and how different components depend on one another. Undocumented dependencies, however, may result in difficulties and issues with our framework during game development; how a developer may spend an unneeded amount of time figuring out how a certain component interacts with others, all the while trying to resolve issues as they appear due to unintended implementation.

5.2.1 Cause of Risk

Inattentiveness: As we make continual changes to our framework, the more changes we must make to the documentation as well. Failure to update the documentation in accordance to the changes made to the framework will result in confusion for future developers.

5.2.2 Likelihood

Moderate: Dependencies can change often; so much so that it may be easy to miss document these changes. This will lead to compatibility issues, making the system less stable. So, it's crucial to stay on top of these changes to avoid problems in our project.

5.2.3 Future Steps

To avoid future difficulties and confusion during game development, we will create a solid dependency tracking system that allows for consistent monitoring of all dependencies via a comparison between such dependencies and their corresponding documentation. Should there be development issues as a result of poor documentation, then it is back to the drawing board with the design documentation; this, of course, will cost unneeded time and resources. This approach management ensures the stability of our system and minimizes the risk of potential struggles during development.

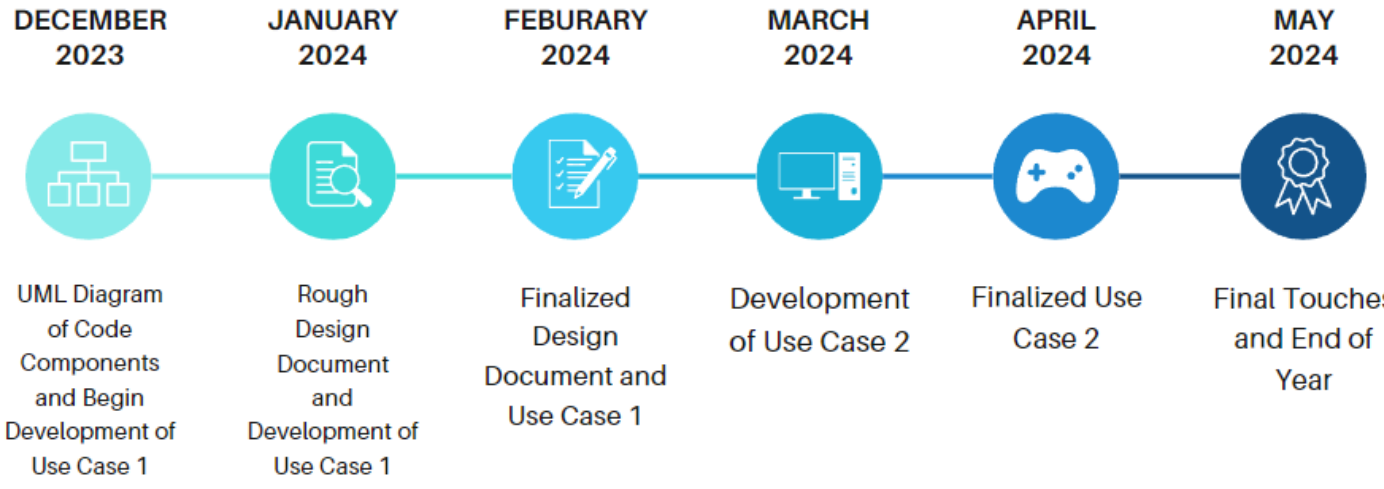
6.0 PROJECT PLAN

As stated in the previous section, our project will span an entire academic year: from the fall semester of 2023, to the spring semester of 2024. Within such a span of time, the following is a list of milestones we have identified thus far.

- **Requirements Specification Document:** This document describes the expected functional and non-functional capabilities the team plans to achieve with the framework. **Completion Date: December 5, 2023.**
- **UML Diagram:** This diagram in particular will be a class diagram showcasing the relationship of proposed code components of our framework; it will be delivered to the client. **Completion Date: November 15, 2023.**
- **Design Review:** This will be a presentation, to the capstone class, communicating what we have accomplished thus far, the current status of the project, and a schedule of the remaining tasks to accomplish. **Completion Date: November 30, 2023.**
- **Technical Demo:** This is a live demonstration, to the capstone class including our client, of a proof-of-concept for how our framework should function. **Completion Date: December 5, 2023.**
- **Two Use Cases:** This refers to the two games that will be developed as examples of our framework, then deployed on the web via hosted on itch.io. **Completion Date: April, 2023**

The following graphic [Figure 6.0] is a timeline chart of the identified milestones.

Figure 6.0 - Timeline: Fall 2023 to Spring 2024



7.0 CONCLUSION

Vaccinations could play a critical role in improving symptoms of viruses, long term effects of illness, and fatalities related to such viruses. Currently, there are no ongoing studies that target adolescent vaccination rate improvements for COVID-19 and HPV. That is where our client, Dr. Amresh, comes in; how he plans to conduct such a study aimed at improving the adolescent vaccination rates through video games. In particular, our client's proposed workflow consists of developing a vaccine literacy burst game, then deploying that game to conduct his studies. The bottleneck in such a workflow, however, is the development process of the video game; how developing a video game from scratch will take up a substantial amount of time and resources. And so, our solution to such a bottleneck is to provide a video game framework in which future developers can easily build off of and extend. Thus far, we have outlined the various functional and non-functional capabilities that our framework is expected to possess; capabilities such as knowledge-drop components, game-mechanics repository, and maintainability. But of course, implementing these capabilities does not come without risks to consider; risks such as our system being computationally intensive leading to a poor player experience, and poor documentation leading to development confusion and issues. We hope that this first, but very crucial step, in our client's plan will result in a significant improvement to the adolescent vaccination rates, and quite possibly end the current pandemic situation.

8.0 GLOSSARY AND REFERENCES

Burst Games: A video game characterized by its quick-paced and engaging gameplay that requires minimum playtime and low frustration levels for players.

Entity-Component System: A design pattern that is mainly utilized in a game development context. This design pattern focuses on composition over inheritance which provides an efficient way to organize code.

Knowledge-Drops: An educational component incorporated within a video game. This could be material presented to a player during gameplay that influences their decision-making or to promote learning.

Game Mechanics: The rules that guide a player through game play and how the game responds to the player's actions. Core game mechanics outline the gameplay experience.

[1] "Children and COVID-19 Vaccination Trends." *American Academy of Pediatrics*, 3 May 2023, https://www.aap.org/en/pages/2019-novel-coronavirus-covid-19-infections/children-and-covid-19-vaccination-trends/.

[2] "Vaccination Coverage by Age." *Arizona Department of Health Services*, 4 Oct. 2023, https://www.azdhs.gov/covid19/data/index.php#vaccination-coverage-byage.

[3] *Super Puzzle Fighter II Turbo.* Arcade, 1996.

[4] *Jetpack Joyride.* IOS, Flash, Android, 2011.