AN EDUCATIONAL INTERACTIVE SIMULATION TO TEACH LIFEGUARDS-IN-TRAINING FIRST AID AND EMERGENCY PREPAREDNESS USING ANIMATION RIGGING AND 3D MODELING

Ethan Chen

Sage Hill, 20402 Newport Coast, 7 Maryland

ABSTRACT

Many people today lack basic first aid knowledge and emergency preparedness. Official first aid training requires too large of a time commitment and lacks concision, causing many learners and lifeguards-in-training to be deincentivized. In order to combat this problem, we created a first aid simulative video game in which the player acts as a lifeguard and needs to carry out various procedures such as CPR and water rescue. We used Unity Game Engine, Visual Studio 20, C# programming, mixamo.com, and the Unity Asset Store to construct our game. A key challenge we underwent was animating the various models due to the complex nature of creating animations. Instead of making the animations ourselves, we used mixamo.com to download already-published animations to import into our own game. In order to enhance the user experience, we experimented with the camera speed and hitboxes in order to get all the details right. Ultimately, Lifeguard Simulator provides a concise, captivating, and enjoyable first aid learning experience that is not available elsewhere.

KEYWORDS

Unity Game Engine, Lifeguard Training, 3D Interactive Simulation, Animation Rigging

1. INTRODUCTION

1.1. Introduction to Problem

Over 85% of today's adults "lack the knowledge and confidence to act if someone was un responsive", leaving them unprepared for everyday emergency situations such as choking and unconsciousness, according to redcross.org.uk. This lack of awareness leads many individuals to take emergency medical services for granted, failing to realize that ambulatory services cannot arrive at the scene immediately. In such instances, essential medical knowledge and skills within the general population are crucial to saving lives. As a certified lifeguard, we recognized the pressing need for more first-aid education, having undergone over twenty hours of rigorous training in first-aid skills, emergency safety, and lifesaving water practices to obtain my certification. It became clear that these vital skills were not widely taught in schools and that learning first-aid required an extensive time commitment. We believe first aid education is practical for all ages, not just adults or older teenagers. To appeal to younger audiences, we needed to create a method in first aid education that was not only concise but also able to captivate attention and not become stale after a while to suit the shorter attention spans in children. According to www.britishcouncil.my, "Children have much shorter attention spans than adults". We saw this as an opportunity to innovate by creating an interactive 3D video game that

David C. Wyld et al. (Eds): NSEC, InWeS, CSIP, BMLI, CACIT, EDUIT -2023
pp. 141-155, 2023. CS & IT - CSCP 2023 DOI: 10.5121/csit.2023.131513

would educate young audiences on critical first-aid skills engagingly and entertainingly. But creating this project would require a lot of brainstorming, prototyping, and obstacles to overcome to make it a fully published game.

1.2. Method Proposal

To address this need, we invested significant time and effort in motion tracking and 3D modeling to innovate an interactive video game that emulates an emergency. We utilized my prior programming knowledge and extensive experience with Unity, a versatile game development software, to create an interactive game that would be unique and stand out from traditional first-aid courses online. We designed it to be a lifeguard roleplay game that would incorporate the same training we went through to become lifeguard certified in the game while making it enjoyable and not as tedious. The video game uses animation rigging, 3D modeling, keyframes, C# code, and Uwe design to enhance the player experience. The game includes many common scenarios, such as spinal injury and CPR. We incorporated various elements from my official lifeguard certification training into the outline of my game. For example, the player must signify an emergency by blowing their whistle, properly position an unconscious victim onto a flotation tube, and use a backboard, which are all procedures we previously learned in my former training. The main thing that differentiates my game from official training is that it is more concise and captivating, meaning younger children who don't want to spend hours going through online quizzes can learn the same maneuvers by playing an exciting and interactive video game on their devices. By publishing my game, we hope to contribute to general first-aid education for the online community, reducing the likelihood of emergency situations and their potential consequences and making the world safer. The following paper is structured in this order: challenges, method analysis, experimentation, comparing methodologies, conclusions, summary, and references.

2. CHALLENGES

2.1. Challenge A

A challenge we faced when implementing the various animations throughout the game was the keyframes. Many of the models were very complex and would take hours upon hours of time to animate smoothly. For example, each limb of a character was its own component that had to be keyframed separately. Also, the game had three scenarios: CPR, rescuing an unconscious victim, and rescuing spinal injury victims. Each one of these three scenarios had around 4-5 different animations. To overcome this issue, my mentor and we used a website called mixamo.com, where we downloaded animation files (.fbx) and imported them into our game. The animations were high quality and worked well, saving us great effort.

2.2. Challenge B

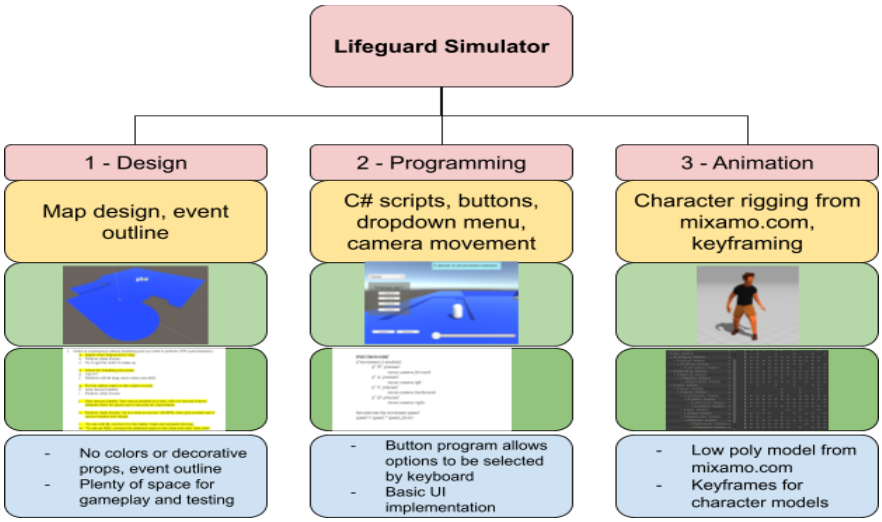
Another challenge my mentor and I faced while working on our video game project was the layout of the pool area where the game would take place in, also called the "map." The map needed enough space for the various events to take place but also not too large to make the environment feel disproportional. To elaborate, most swimming pools are 20 feet wide and 40 feet long, and we had to recreate those dimensions in a modeling engine that did not have an easy way to measure feet. To navigate this problem, we used the character models and assumed that the width was around 1.5 feet; using this measurement, we could construct a realistic-looking pool environment with similar, if not perfect, dimensions.

2.3. Challenge C

A third challenge my mentor and we encountered while developing the lifeguard game was the camera movement. While brainstorming the overall layout and functionality of the player experience, we wanted the camera to revolve around the player and follow the player while still being rotated by the player's mouse movements. To do this, we had to assign the camera to the player's avatar. However, the camera was locked in place and would not rotate according to mouse movements. we developed a c# script in Visual Studio that assigned an invisible sphere around the player character to fix this issue. As the character moved, the sphere moved with it, and the camera was set to the sphere. The sphere would move with the lifeguard character as the player used WASD, while the camera would move around the sphere with the movement of the mouse.

The performance of this application is heavily impacted by the artificial intelligence model that we are using. We must first consider where we are going to find adequate data, as Awe models need hundreds of images to train. To expedite this process, we will use an image scraper on Flickr. The Awe must also be comprehensive, but it cannot be so large that development would take too long, but not so small that it serves no utility. We will establish that at minimum our Awe must be able to identify basic foods such as bread and rice. Our Awe must also be accurate at identifying these objects to provide utility to users and so they are not frustrated. To account for this, we will train at least 500 images per object and perform quantitative assessments for adjustment.

3. METHOD ANALYSIS



3.1. System Overview

Three significant components that my program links together are the design, programming, and animation elements.

The design segment consists of the basic map design and event outline. The map design was very primitive and straightforward at first - just for my mentor and me to get a general sense of how the pool environment would look. After getting the dimensions right, we then added the decorative props to add some flair and life to the map. We based the layout on a community pool in Southern California, with a slope, kiddie zone, lifeguard stand, ramp, deep end, beach chairs,

and palm trees. The event outline was a document listing all the events that would take place during the game. We consistently used this outline sheet for referencing during the programming stages.

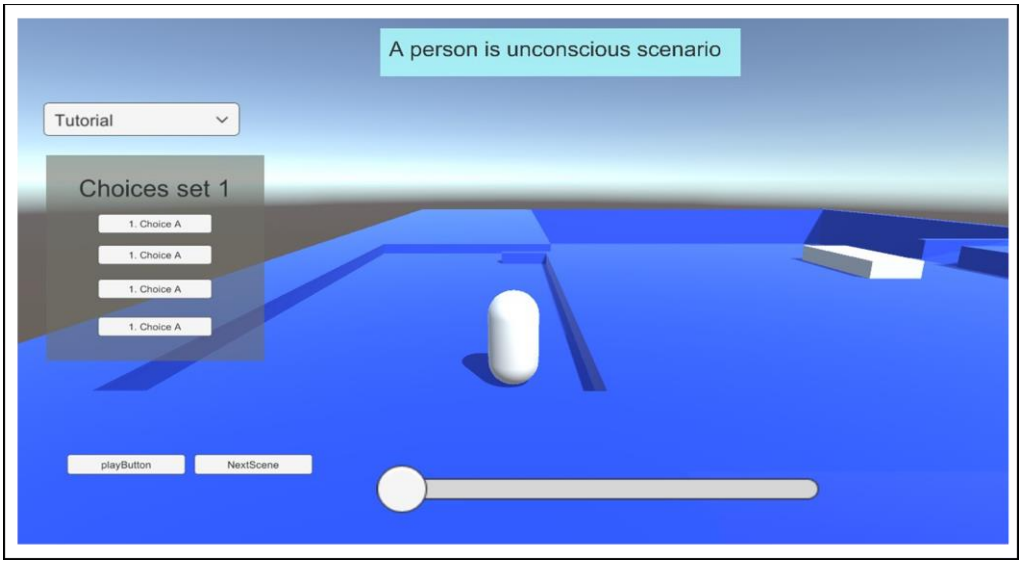
The programming aspect of my game consisted of C# scripts written in visual studio, buttons, dropdown menus, and camera movement. The Uwe development was primarily done in Unity Game Engine, with certain parts like the start and end menu made in Photoshop. Most of the scripts used elements such as if statements, loops, variables, and external libraries.

The final aspect was the animation element of my lifeguard simulator. This branch consisted of animation rigging and .fbx files from mixamo.com, keyframes, and animator controllers. Animation rigging essentially assigns various plot points on a 3D character model, and that plot points to control a particular part of that model (such as an arm or leg). They are controlled with keyframes, which are a set of directions given to the rig to move certain areas at certain times. Animator controllers then control which animations play in which scenes or events in the game.

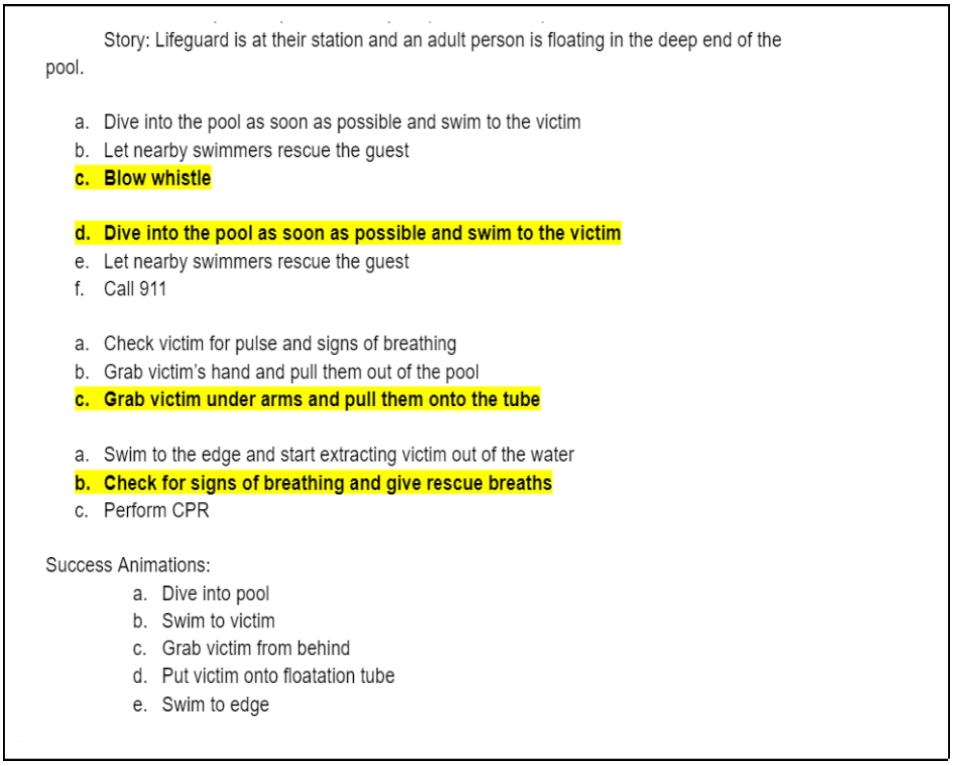
3.2.1. Component Analysis A

The purpose of the design component was to outline how the game would play out and what the user experience would look like. In Lifeguard Simulator, an animation plays, and the user has to answer questions according to the animation that relate to first aid procedures. The questions and the answers (with the correct one being highlighted) were all listed in the document and helped me develop them in the game engine. The flow of events was validated by playtesting from various people of gaming or first aid backgrounds. The order and occurrences of the many scenarios was revised many times from the original outline.

3.2.2. Uwe Screenshot



3.2.3. Code Sample



Story: Lifeguard is at their station and an adult person is floating in the deep end of the pool.

a. Dive into the pool as soon as possible and swim to the victim
b. Let nearby swimmers rescue the guest
c. Blow whistle

d. Dive into the pool as soon as possible and swim to the victim
e. Let nearby swimmers rescue the guest
f. Call 911

a. Check victim for pulse and signs of breathing
b. Grab victim's hand and pull them out of the pool
c. Grab victim under arms and pull them onto the tube

a. Swim to the edge and start extracting victim out of the water
b. Check for signs of breathing and give rescue breaths
c. Perform CPR

Success Animations:

a. Dive into pool
b. Swim to victim
c. Grab victim from behind
d. Put victim onto floatation tube
e. Swim to edge

3.2.4. Code Explanation

The above image shows the flowchart we used to outline the events that would take place throughout the game. It was essentially the "story" of the simulation and helped me understand the amount of content and flow of events. Because we had to manually input each question and answer into Unity Game Engine, and then select which one of the three answers would be marked as correct, it helped a lot to have the questions laid out for me with the correct answers highlighted. The list of animations at the bottom allowed me to know which animation would play after another. Knowing the order of animations was very helpful and saved a lot of time with positioning and setting up the keyframes properly. For example, if the diving animation is before the swimming animation, then we know that the swim animation should start in the pool at the same spot the diving animation finished.

3.3. Component B

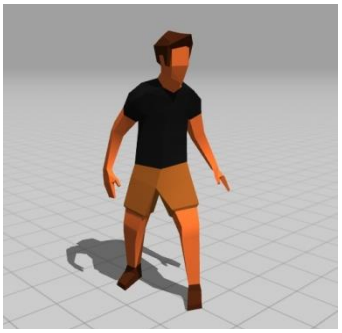
The purpose of the programming component is to give the game its functionality. The C# scripts my mentor and we enhanced the user experience and allowed him/her to be able to do various things such as: to control the camera, to view the animations, and to answer the in-game questions.

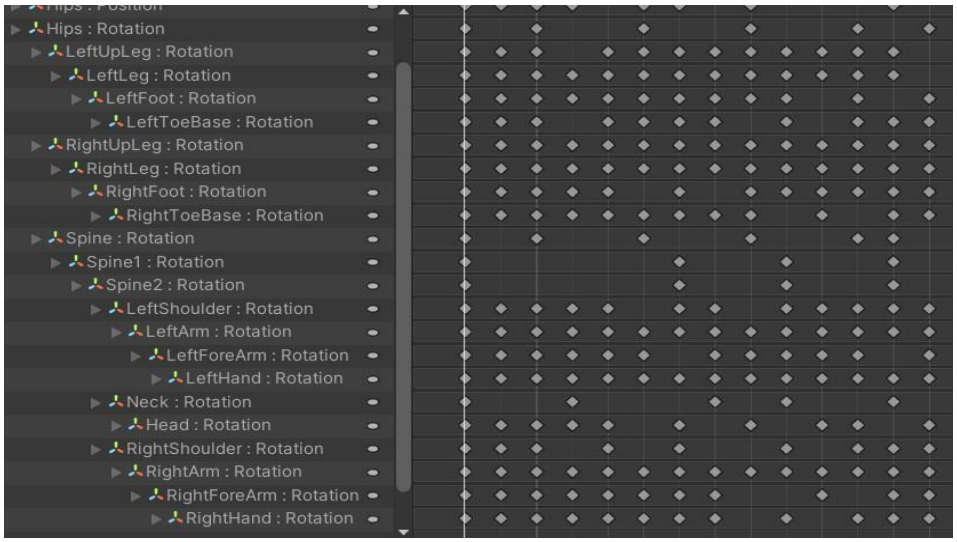
```
PSEUDOCODE
if movement is enabled:
        if 'W' pressed:
                move camera forward
        if 'A' pressed:
                move camera left
        if 'S' pressed:
                move camera backwards
        if 'D' pressed:
                move camera right

#accelerate the movement speed
speed = speed * speed_factor
```

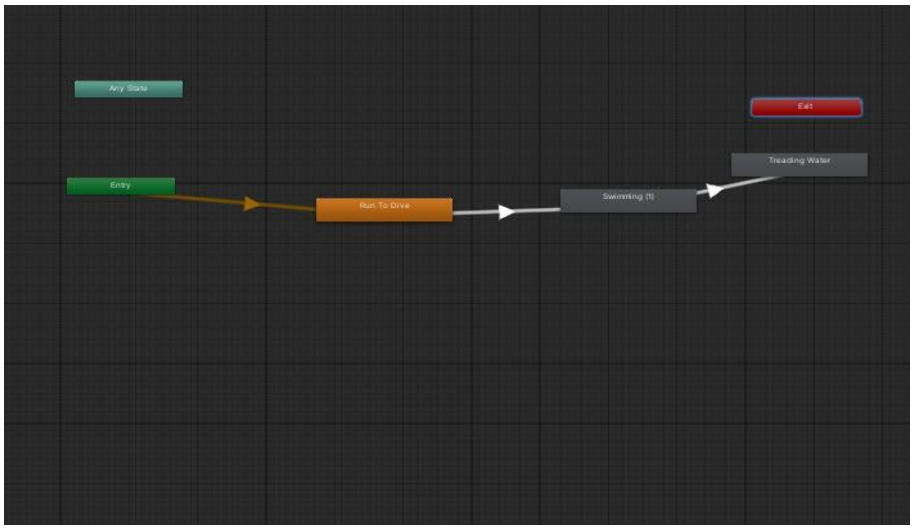
This screenshot is pseudocode of a C# script developed in Visual Studio 2021 and then inputted into Unity Game Engine. The nested if statement essentially turns the user's keyboard inputs, such as WASD, into a movement for the camera. My mentor and we developed this movement during the earlierd stages, before finishing the map development, in order to get a basic idea of what the game was going to look like. Since the game is a role-playing game with set animations, the user cannot control the player, they can only control the camera to rotate around the environment and get a better view of the animation that plays out (which could be CPR, extrication, water rescue, etc). The "calculate acceleration" segment at the bottom of the screenshot depicts how we multiplied the current speed value with a separate value to exponentiate the speed of the movement, so that the camera would gradually increase from motionless to moving, rather than suddenly moving and then stopping without any acceleration or deceleration.

3.4. Component C

The animation component of Lifeguard Simulator is the foundational basis for all the movements happening in the game. Keyframes, rigging, and animator controllers dictate various functions in the game such as the character's running, swimming, diving, etc. Many of the animations were imported into Unity from a development website called mixamo.com.







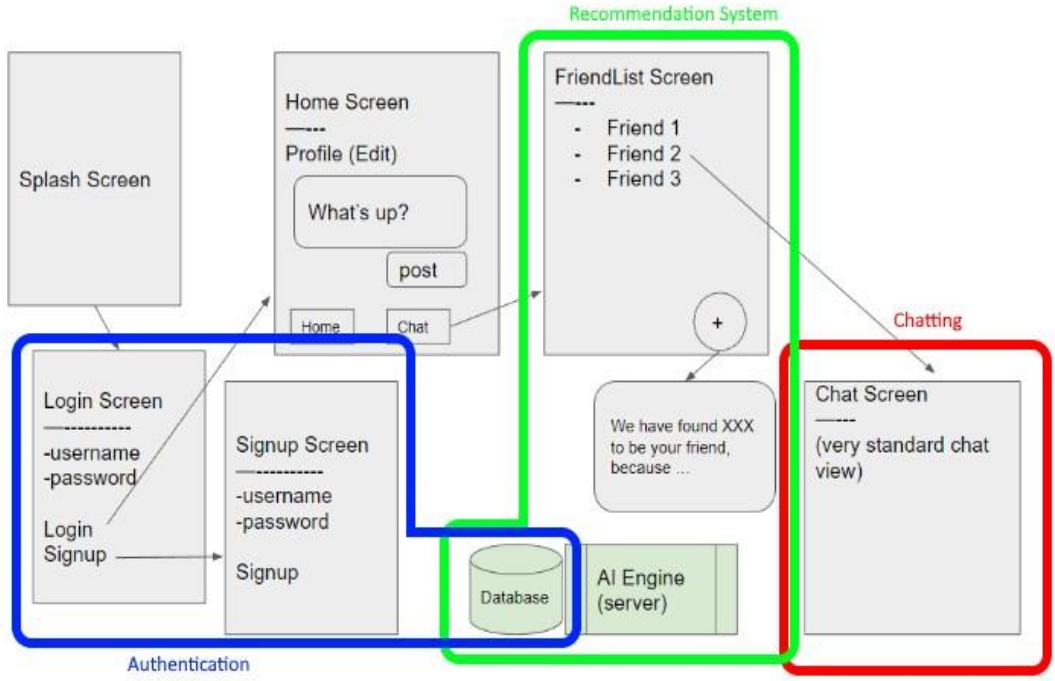
The 3D rig in Figure 4 was used as the player. After replacing the materials (colors) and making a few adjustments to the size, we added various animations to the player model using Unity's keyframing system. Figure 5 shows the keyframes that were used in the diving animation for the lifeguard character. The various dots on the right hand side are the keyframes, which are certain points in time that control where a certain part of the rig will move (for example, a keyframe could be used to make the left hand go upwards 0.5 seconds after the animation has been started). To save time and avoid meticulously plotting out each keyframe, this animation, as well as other animations on the more complex side were imported externally. Figure 5 depicts an animator controller, which acts as a flowchart and dictates when a certain animation to play. The green box labeled "entry" is the starting point of the controller, which then moves to the right according to the arrows and ends on the "Treading Water" animation.

3.5. System Overview

Main Structure:

- Three parts: an authentication service, a recommendation system, and a chatting system
- Information is all uploaded and stored to Firebase database

Here's our flow chart. Let's circle the 3 major components that we can identify. For 3.1A, you would put the flow chart without these circles.



Flow is as simple as following the arrows.

- Get splash screen
- Login Screen
- Signup Screen if needed
- Both go to the dashboard
    - you can make posts from the dashboard
    - displays profile information
    - Friend Screen
    - shows all your friends that you can talk to
    - if you don't have any, then ask for a recommendation
    - Awe server will respond back with a potential new friend
    - Chat Screen
    - chat with someone
    - similar to SMS or

WhatsApp What did you use to make this program?

- Flutter
- Firebase
- Repl.it

The program is split into three large components: an authentication service, a recommendation system, and a chatting system. All the information sent through the app is stored within a database to later be processed by a sentiment analysis AI. To construct this program, we used Flutter for the codebase, Firebase for authentication and database services, and Repl.it to host our sentiment analysis Awe model. We decided to use Firebase primarily because of its ease of use but also because we can allow certain portions of our database to be accessible through HTTP

requests. We decided to host our sentiment analysis Awe on a repl.it server running flask because it is simple to set up and make APwe requests to. Our program is designed to be a simple application that users can download free on app stores. When opened, it will display a splash screen to the user and prompt them to log in. If the user doesn't have an account, they may sign up. Logging in and signing up both use the authentication service offered by Google's Firebase, which allows us to manage a database efficiently. When they log in they are shown the dashboard, and can make posts that are uploaded to Firebase, or they can visit a Friends List screen. If they do not have friends in the system, they can prompt the app for a friend. The app will send an HTTP request to a server hosting a sentimentality Awe that will determine the best match for a user. There is also a chat screen for users to communicate through that operates similarly to SMS or WhatsApp messaging.

3.6. Recommendation System

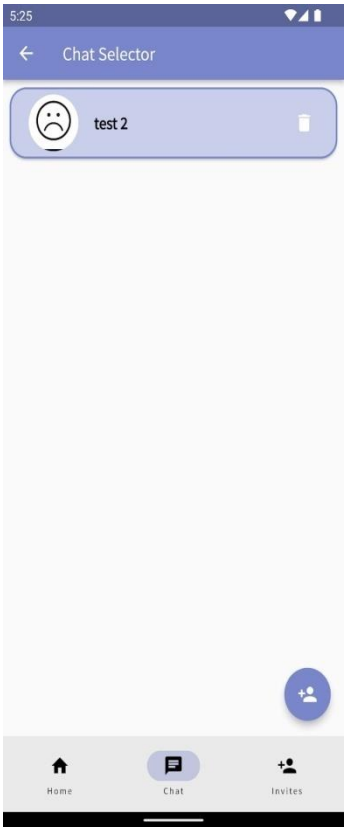
Let’s try writing about the third component.

- What is its purpose?
    - the “social” component of the app, allows users to connect with one another
- What services did you use?
    - Repl.it server
- Explain special concepts
    - Natural Language Processing
        - a way of computing language as data
        - ability for machines to recognize human emotions and suggestions through written words only

- Connect to program
    - To be used in conjunction with friend request system to find new users to chat with, and is required in order to instigate a conversation

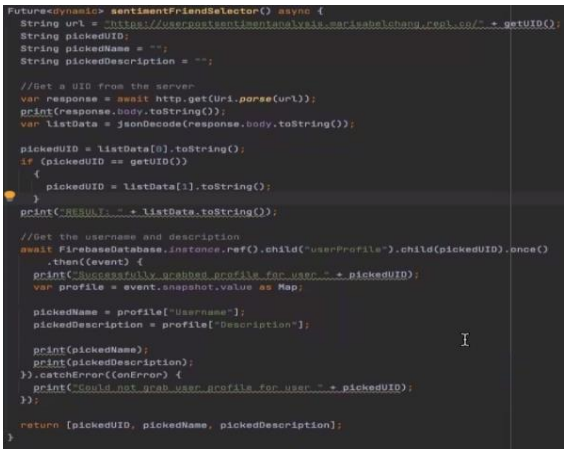
A key part to the app’s functionality is our recommendation system. This system is integral to the app’s core design because it is the primary way for users to find new users to send friend requests to. This recommendation system relies on a simple repl.it backend server, that hosts an NLP model. NLP, or Natural Language Processing, is a way to compute written language as data and recognize human emotions and suggestions from that data.

3.7. UWE Screenshot



3.8. Screenshot

We need to find a screenshot that best represents the component and best illustrates how the component is used within the program's codebase. The method that we decided to show here, sentiment Friend Selector, includes the code where we make a request to a server, and shows how we process the information that we get back.



3.9. Code Explanation

- When does it run in the program?
    - On a friend recommendation page, when the user presses a recommend friend button
- What does the method represent?
    - sentimentFriendsSelector will make an HTTP request to the server hosting the NLP AI, and then display to the user the profile information of the user that the server sent back
- What variables are being made?
    - url – the URL of the HTTP request. To request the server for a recommendation for a specific user, we attach their unique identification at the end of the URL.
    - pickedUID – the user identification of the user that the server sends back. Used to lookup the user's profile information ● What does the server do
    - Awe model will go through all user's posts and perform sentiment analysis on each post that they've made
    - It computes the average sentiment per post for that user
    - It does the same for every other user
    - It returns the UID of the user whose average is closest to our user
- Step by step process
    - 1. Generate a URL with the user's UID.
    - 2. Make an HTTP request with that URL. Wait until response
    - 3. With UID, look up Firebase Database and get corresponding user profile info
    - 4. Return an array with the UID, name, and description
    - 5. Data is shown as a pop up and user can either add them as a friend or ask for new recommendation

The method *sentimentFriendSelector* best represents this component. It is called in the program on a recommended friend page when the user presses a button. When called, it will make a request to a separate server hosting an NLP model, and it will then display to the user the profile information of a recommended friend. There are two important variables. "url" is the URL of the HTTP request. To request the server for a recommendation for a specific user, we attach their unique identification at the end of the URL. "pickedUID" is the user identification of the user that the server sends back. It is used to lookup the user's profile information. When the method runs, it first makes an HTTP request with the URL. The method waits until the data is sent back. Serverside, the Awe model will go through all user's posts and perform sentiment analysis on each post that they've made. It computes the average sentiment per post for that user, with lower scores representing more negative posts and higher scores representing more happy posts. It does the same for every other user and returns the UID of the user whose average is closest to our user. Once the server sends a UID back, we search our Firebase Database in the userProfile section and use that UID to access the profile information of the recommended user. Once all the data is compiled, we wrap it up in an array and return it. In the program, this data is then shown to the user via a pop up and the user is given a choice to either request a new recommendation, or accept it, and send a friend request.

4. EXPERIMENTS

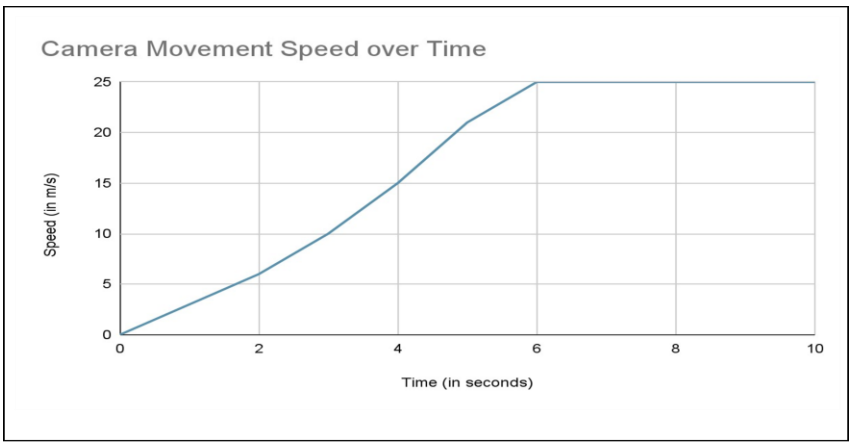
4.1. Experiment A

A possible blind spot in our program is the camera being moved too far from the rest of the game environment. We hypothesized that if the map goes out of view, the player might not be able to return back to the map.

4.2. Design

To set up an experiment and test this bug, we exported an early prototype of the game in a .exe format from Unity Game Engine and ran it as if it were the finished product. We would then move the camera as far away from the map as possible using the WASD keys, and, according to our hypothesis, the map would go out of view and the player would not be able to return. If this were to happen to an actual player it would become problematic because then they would not be able to see any of the in-game animations.

4.3. Data and Visualization



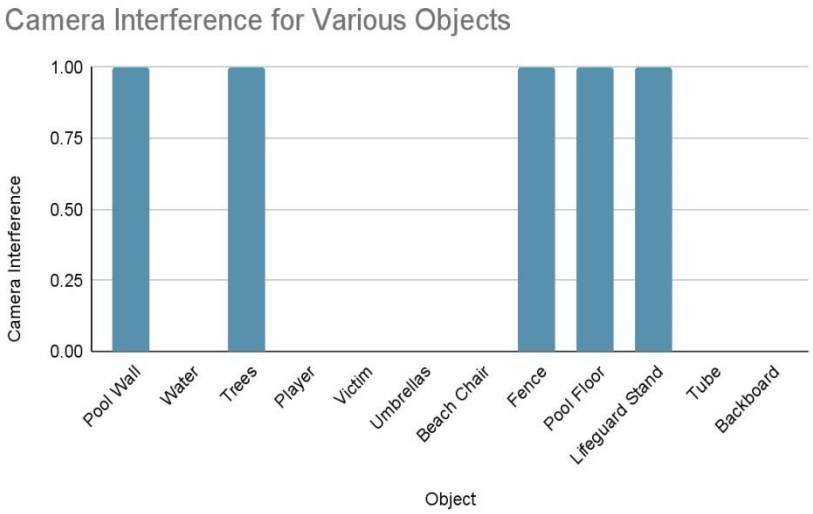
4.4. Analysis

The lowest speed value in the graph is zero because the camera was initially not moving during the experiment. The average speed over the 10-second time frame was 16.4 meters per second, and the median was 21 meters per second. The highest value, or maximum value, in the dataset was 25 meters per second. In the graph, we found out that the camera speed increases exponentially for the first 6 seconds, and then it maxes out at 25 meters per second. This means that the player could accidentally accelerate the camera while trying to make small movements because of the exponential speed increase and accidentally move away from the map. To fix this, we implemented a C# script that would reset the position of the camera after every animation (question answered using number keys). This way, the player is unable to lose sight of the map because of the position resetting.

4.5. Experiment 2

In order to test this experiment, we decided to run our executable prototype again, but this time we tried to move the camera “inside” the one of the walls within the pool. In order to allow for more time to test which objects could and could not interfere with the hitbox of the camera, we

temporarily disabled the animation scripts so that the program would just be a map sandbox for testing the camera movement. We also got rid of certain items that were unnecessary or overly excessive in order to simplify the map and make our job slightly less tedious.



In the chart, a number 1 depicts that the object interfered with the camera in some way, and a number 0 means that the object did not interfere (e.g. the camera just passing through the object or around it). For the objects that did interfere, the camera would become glitched and show a distorted version of the map, as well as become stuck inside the object. A trend that we noticed from looking at this data was that the items with smaller hitboxes would not mess with the camera, such as the umbrellas and beach chairs, but the objects with larger hitboxes, such as the pool walls and the trees, were trapping the camera. In order to fix this, we decided to shrink the hitboxes so that there was not enough space for the camera to fit inside and become trapped in. Since the majority of the game runs on pre-determined animations, super-accurate hitboxes were not necessary for the playability of the game.

5. METHODOLOGY COMPARISON

5.1. Methodology A

Another solution that also solves the problem of lack of first aid education is the Red Cross Training organization. Their online website helps navigate learners to nearby classes based on their location or give them an online training course opportunity that includes “instructor training and bridging” (Red Cross). This solution is reasonably effective but is still very tedious and will not captivate the attention of a young child. It also requires a sizable time commitment, taking around 10-20 hours of training time, while our Lifeguard Simulator only takes up to 30 minutes. Some of the things it ignores are ways to make the training more exciting and appealing to younger audiences, which the simulator does more effectively by incorporating the same training fundamentals into a video game format.

5.2. Methodology B

Another solution to the problem of teaching first aid to younger audiences is a video game called PWA Protocol which is very similar to our Lifeguard Simulator in which it also aims to teach first various first aid procedures to younger audiences by taking first aid training basics and

presenting it in a fun and exciting way in order for them to "develop new problem-solving skills" (MDPI 1). PWA Protocol is very effective at making training visually appealing and similar to other video games such as Crossy Road. However, PWA Protocol is mainly focused on more basic emergency scenarios while Lifeguard Simulator focuses more on pool emergencies, such as drowning, water rescues, and extrication.

5.4. Methodology C

RED CROSS First Aid Learning is a similar first aid educational program that features a more cartoonish presentation in order to appeal to younger audiences and captivate the attention of children. It is quite effective at teaching proper procedures for common occurrences and how to "recognize the everyday hazards in different life situations" (Gameplays). However, it is not very interactive, as it simply shows animations that you click through and scroll by on your device. It would be difficult for a child to retain all of the information just from seeing one animation. Lifeguard Simulator is much more interactive because the player has to click the correct action option and is also immersed in a 3D simulative environment.

6. CONCLUSIONS

6.1. Limitations and Improvements

A limitation to Lifeguard Simulator is that the number of emergency situations is still somewhat small. There are only three different scenarios: CPR, water rescue, and extrication. We believe that some more situations should be added to the project in the near future in order to provide more content for the player to learn as well as a more extensive playing experience. Some situations we have considered but not yet implemented are: splints, emergency oxygen, and Heimlich manuever, which are all lesser known procedures that are still important to know if the need arises. We are also considering a "challenge scenario" at the end of the game where the pool is filled with bystanders and the player (the lifeguard) has to find anyone that needs rescuing and save them in time. Another limitation is that the map environment is still somewhat basic, so we are also considering adding more decorative items and bystanders to the map in order to provide a more immersive experience for the player.

6.2. Concluding Remarks

Overall, this project has been an invigorating and meaningful experience and has taught us various things about video game development such as programming and animation, as well as publishing and marketing. We believe it has truly been a rewarding experience seeing the game come together and our hard work paying off.

7. SUMMARIES

7.1. Experiment Recap

After playtesting the game, we found two possible blind spots in our program that we wanted to fix: the camera being moved too far from the game environment and the camera being stuck inside an object. We set up experiments for each of these issues by exporting an early prototype of the game in a .exe format from Unity Game Engine and ran it as if it were the finished product. We found out that the camera speed increased exponentially and was causing difficulties for the player to control the camera's movement. To fix this, we made the camera teleport back to the starting point after every animation played. To fix the issue of the camera being stuck inside an

object, we collected data and figured out that larger hitboxes were causing the problem. To fix this issue, we shrank the hiboxes so that they were too small for the camera's hitbox to fit inside of.

7.2. Methodology Comparison

The first methodology that solves a similar problem is the Red Cross Training platform. The website navigates aspiring learners to nearby classes or online courses and makes it much easier for people to find the proper resources that they can learn from. However, the platform is not very interesting or exciting for younger minds who do not have a very extensive attention span. The second methodology is a first aid video game called PWA Protocol which, like Lifeguard Simulator, provides basic teaching of emergency procedures in the form of an interactive game. However, it is focused on already well-known situations in which their proper procedures can easily be found online. Lifeguard Simulator teaches lesser known procedures like extrication and water rescues. The third methodology is a game called RED CROSS First Aid Learning. This app is specifically designed for younger audiences with a cartoonish aesthetic using 3D interactive simulations and animation rigging. However, it is not very interactive and a child may have difficulty retaining the information presented in this app.

REFERENCES

[1] First aid: only one in 20. (n.d.). British Red Cross. https://www.redcross.org.uk/first-aid/only-one-in-20British Council. (2019). How to Increase Your Child's Attention Span. Britishcouncil.my.https://www.britishcouncil.my/english/courses-children/resources/attention-span#:~:text=The %20first%20thing%20to%20know

[2] American Red Cross.https://www.redcross.org/take-a-class?scode=PSG00000E017&&&&&ds_rl=1289062cid=generator&med=cpc&source=google&gad=1&gclid=Cj0KCQjwy9-kBhCHARIsAHpBjHh-VPHO30wjO7SHkr0buRaUVShZI39t9utHri8Spjz8_l17KDLGal8aAip_EALw_wcB&gclsrc=aw.ds

[3] Cristina Rebollo. (2021). Learning First Aid with a Video Game. MDPI.https://www.mdpi.com/2076-3417/11/24/11633

[4] Gameplays TV. (2017). RED CROSS First Aid Learning Video Game for Kids. Youtube.com.https://www.youtube.com/watch?v=RGndpkpeTY8