AttentionSense: Using Extended Reality to Detect the Levels of Attention in Situational Context for ADHD Identification

Final Report

Arda Bakici - arda.bakici@nyu.edu

Muhammet Mustafa Diri - mmd9622@nyu.edu

Ray Verma - rv2340@nyu.edu

Video Demonstration of the Project:
https://drive.google.com/file/d/1xOMBOmn_WTZraDx2WRsatgZnuDqtySqC/view?usp=sharing

Github repository where project code is stored: https://github.com/ArdaBakici/CPE-Project

1. Introduction

1.1 Introduction to ADHD

ADHD (Attention-Deficit/Hyperactivity Disorder) is one of the most common mental disorders, affecting decision making skills, long-term focus, and hyperactivity. This further leads to difficulties in academic, professional, and social settings [1]. Diagnosis of ADHD should be made by a qualified psychiatrist and requires a comprehensive assessment, including physical examinations and multiple interviews. This process requires several visits to the psychiatrist, which can be time consuming and expensive, while being inaccessible in many parts of the world. There is a need for a quicker, cheaper, and convenient method [2]. In order to solve these problems, we plan to apply Virtual Reality to help with ADHD Diagnosis.

While trying to focus on a task, people with ADHD showcase a high frequency of involuntary visuomotor reflexes, as well as a greater latency while responding to visual stimuli [3]. In addition to evaluating these metrics, responding to questions that test memory and attention throughout the task would further increase the chances of the correct diagnosis.

Using these criteria, this project allows for the preliminary diagnosis of ADHD, saving the time of both the psychiatrist and the patient. The scope of this project is defined for children and teenagers, since symptoms of ADHD start to occur and show themselves in this period. Though adults may also utilize the application, the results may not be conclusive and will require extra sessions with the psychiatrist to confirm that the symptoms are present since childhood [2]. Additionally, real-world implementations of this product might be possible under the supervision of authorities.

1.2 Project Objectives

The objective of this project is to make a VR application that can accurately identify symptoms of ADHD in adolescents using the Oculus Quest 2 headset. The application should capture and analyze head-tracking data and gauge user attention during the narration of a story in a stimulus-rich environment for assessing the chances of a positive ADHD diagnosis, and showcase these metrics for statistical evaluation through user-friendly visualizations.

This application aims to do the following:

- Reduce the time taken for diagnosis: Using the application, preliminary symptoms of ADHD should be identified within 10 minutes, thereby reducing the time taken in a long (1 hour) formal assessment.
- Reduce the cost of diagnosis: Due to the removal of the need of a psychiatrist, the estimated cost of evaluation is reduced. This initial assessment will help determine if a formal diagnosis is required, only then will a patient require a medical professional.
- Make diagnosis accessible: In parts of the world where ADHD diagnosis is inaccessible, this preliminary assessment may be obtained easily, allowing for the easily-obtainable identification of the disorder in a patient.

2. Project Development

2.1 Setting of the Narrative

2.1.1 Story

“The world is changed. I feel it in the water. I feel it in the Earth. I smell it in the air. Much that once was is lost. For none now live who remember it. It began with the forging of the great rings. Three were given to the Elves, immortal, wisest, and fairest of all beings. Seven to the Dwarf lords, great miners, and craftsmen of the mountain halls. And nine, nine rings were gifted to the race of Men — who above all else, desire power.

For within these rings was bound the strength and will to govern each race. But they were all of them deceived, for another ring was made. In the land of Mordor, in the fires of Mount Doom, the Dark Lord Sauron forged in secret a master ring, to control all others. And into this Ring, he poured his cruelty, his malice, and his will to dominate all life. One Ring to rule them all.

One by one, the free lands of Middle-Earth fell to the power of the Ring. But there were some who resisted. A last alliance of Men and Elves marched against the armies of Mordor, and on the slopes of Mount Doom they fought for the freedom of Middle-Earth.

The victory for the alliance was near, but the power of the Ring could not be undone. Sauron used his ring to crush all life beneath him. It was in this moment, when all hope had faded, that Isildur, son of the King, took up his father's sword and gave an end to the malice of Sauron. The enemy of the free peoples of Middle-Earth, was then defeated.”

2.1.2 Questions

<u>Q1) Where was the master ring, the One Ring, forged?</u> | Difficulty level: Easy
a. In the land of Elves
b. In the fires of Mount Doom
c. In the peaceful Shire
d. In the land of Men
Answer: b. In the fires of Mount Doom

Q2) Who led the last alliance that marched against the armies of Mordor? | Difficulty level: Medium
a. The Dwarves
b. The Dark Lord Sauron
c. The race of Men
d. Men and Elves
Answer: d. Men and Elves

Q3) What did Isildur do when all hope had faded? | Difficulty level: Medium
a. He surrendered to Sauron
b. He took up his father's sword and defeated Sauron
c. He sought the help of the Dwarves
d. He destroyed the One Ring
Answer: b. He took up his father's sword and defeated Sauron

Q4) What sensory experiences suggest that the world is changing? | Difficulty level: Hard
a. Hearing it in the wind
b. Seeing it in the sky
c. Smelling it in the air
d. Tasting it in the food
Answer: c. Smelling it in the air

Q5) What is the ultimate goal of the One Ring mentioned? | Difficulty level: Medium
a. To bring peace and harmony to Middle-Earth
b. To grant immortality to its bearer
c. To control all other rings and dominate all life
d. To unite the races of Middle-Earth
Answer: c. To control all other rings and dominate all life

2.2 Assets

Resource	Type	Link
The Environment	3D Model	https://open3dlab.com/project/33444/
Dark Knight (The Narrator)	3D Model	https://sketchfab.com/3d-models/dark-knight-e2208bdc46304f6faa18728778986f35
The Crystal Ball	3D Model	https://sketchfab.com/3d-models/crystal-ball-6018233cc2434f8eba72f474657b65c1

The Ring	3D Model	https://sketchfab.com/3d-models/the-one-ring-lord-of-the-rings-39eb401be92c49d39520fadd5ecff8d3
Maliketh's Black Blade	3D Model	https://sketchfab.com/3d-models/maliketh-black-blade-2a05bdc774cb46daaae3b04a606b456c
The Eye of Sauron	3D Model	https://sketchfab.com/3d-models/the-eye-of-sauron-lord-of-the-rings-fadecd2a323b47dab09f7e27402e521d
Mountains	3D Model	https://assetstore.unity.com/packages/3d/environments/high-quality-free-snowy-mountain-game-ready-233788
Table	3D Model	https://sketchfab.com/3d-models/wooden-table-game-ready-asset-7283ac1841504452b53005b8103bb064
50mm Zoom Flare and Flare Small	Lens Flare	https://github.com/jamschutz/Unity-Standard-Assets (Original asset store package got deprecated)
Starfield Skybox	Skybox	https://assetstore.unity.com/packages/2d/textures-materials/sky/starfield-skybox-92717
Blacksmith Sound Effect	Sound	https://www.youtube.com/watch?v=4LXZlwhOLpU
Thunder Sound Effect	Sound	https://www.youtube.com/watch?v=T-BOPr7NXME
Sword Falling Sound Effect	Sound	https://www.youtube.com/watch?v=9bhov4Xolio
Soldier Footstep Sound Effect	Sound	https://www.youtube.com/watch?v=41stu7d1Wok
Army Marching Sound Effect	Sound	https://www.youtube.com/watch?v=J4rUWxyL40k
Background Music	Music	https://www.youtube.com/watch?v=J93fc--VsaI

2.3 List of Animations Used

Animation Name	**Used Where**	**Source**
Neutral Idle	Beginning	Mixamo
Pointing	Lightning	Mixamo
Move Character	Walking from Window to Chair	Custom Made
Right Turn 90	Walking from Window to Chair	Mixamo
Walking	Walking from Window to Chair	Mixamo
Stand to Sit	Sitting	Mixamo
Talking	The Narration	Mixamo
Sitting	The Narration	Mixamo
Ring Flicker	Ring Distraction	Custom Made
Increase Light and Decrease Light	Sauron Distraction	Custom Made
Struck In Head	The Narration	Mixamo
Angry Gesture	The Narration	Mixamo

2.4 Model Rigging and Animations

We got the 3D model for the narrator from Sketchfab. However, this model was not rigged, so we first modified the model in Blender to make it suitable for rigging. For the narrator model, we removed the sword and cloak, and also changed the shape of the left hand. Then we uploaded the model to Mixamo for automatic rigging. Then we downloaded the model and the animations we wanted to use from Mixamo and imported them into Unity.

We modified some of the animations we took from the Mixamo to suit our needs. For example, the 'Struck in the Head' animation was originally for a character getting hit in the head and bringing his hand to the impact area. We trimmed the animation, slowed it down, and disabled some of the bone movement to create a sadness and hopelessness effect.

2.5 Offsets, Extrapolation and Override Layers

We used Unity's timeline system to sequence the animations. However, we encountered a problem when combining them. When a Mixamo animation ended and another animation started, the new animation started from the origin position instead of the position where the last animation left off. This happened because the root motions of the Mixamo animation did not apply to the rigged model. So instead we used extrapolation, track offsets and override layers to combine the animations. In the Unity timeline system, extrapolation refers to preserving the final state of an animation until the next animation starts. However, when the next animation starts, it starts from the original position, not the extrapolated one. So we used track offsets to correct this. Track offsets apply position and rotation offsets to each animation on that track, and animation offsets do the same for a single animation. All of our animation tracks have track offsets to account for the fact that the narrator is not at the origin. For the animation offsets, we combined the ‘Right Turn 90’ animation and the walking animation with the rotational animation offset. Since ‘Right Turn 90’ animation does not change the original rotation of the narrator, we apply rotation offsets to the animations to make them start from the last rotation position where the animation ended.

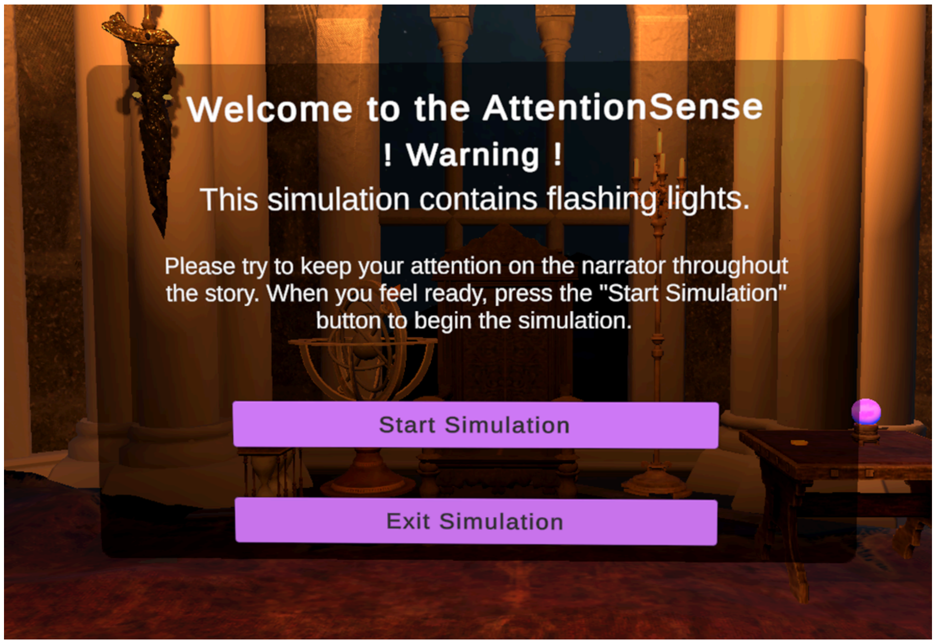
The sitting animations in Mixamo don't have the same position. So when we combined the animations, the movement was glitchy. So we used override layers to combine these animations. In the Unity timeline system, the animations that are on the override layer apply on top of the animation that is on the main track. So the properties that the animation is not altering remain at their current values, unlike normal animations, where even if the animation is not altering the values, they return to their default values. So by combining animation extrapolation and override layers, we can achieve animation movement on the upper body while keeping the legs and sitting position in the same position as the last one, in this case the Stand to Sit animation.

Since we're not using the root motion from the Mixamo animations, we need to move the narrator with another animation while the walk animation plays. For this we have the MoveCharacter animation, which changes the character's z-coordinate from 0 to 3.68 over the span of 5.5 seconds.

We also have other animations such as Ring Flicker, Increase Light, and Decrease Light, which alter light brightness and lens flare intensity properties over several seconds.

2.6 Animated Sequence

The simulation is set in a wizard's study room. The user sits in a chair in front of the narrator. The simulation first greets the user with a menu with 2 options, "Start Simulation" and "Exit Simulation". When the user presses Start Simulation, the menu is disabled and the timeline begins to play.



Welcome to the AttentionSense

! Warning !

This simulation contains flashing lights.

Please try to keep your attention on the narrator throughout the story. When you feel ready, press the "Start Simulation" button to begin the simulation.

Figure 1. Main Menu

The simulation starts with the narrator looking out of the window in idle animation. The background music also starts to play. The user has about 10 seconds to look around, analyze the room and satisfy their curiosity. Then the simulation begins with the first event.

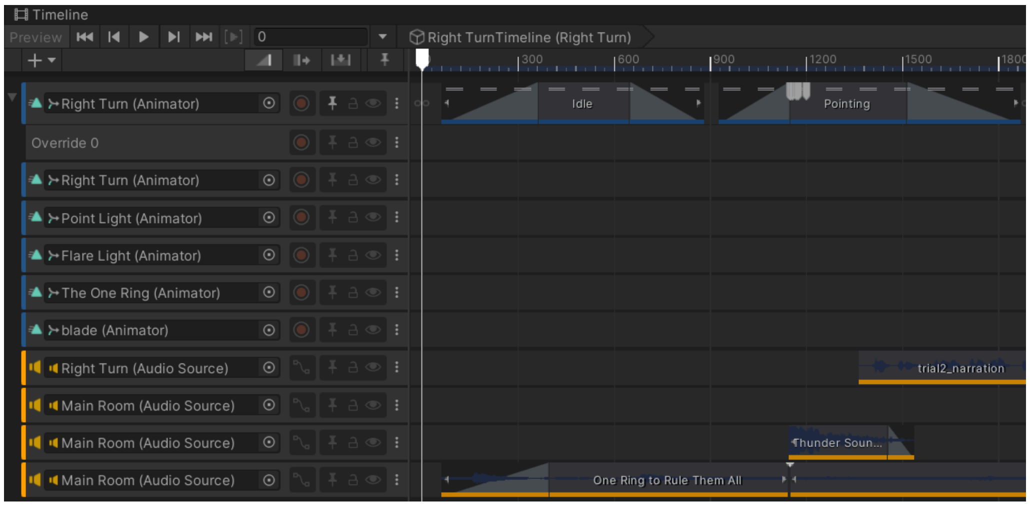


Figure 2. Timeline for the Start and the Lightning Event

2.6.1 Lightning

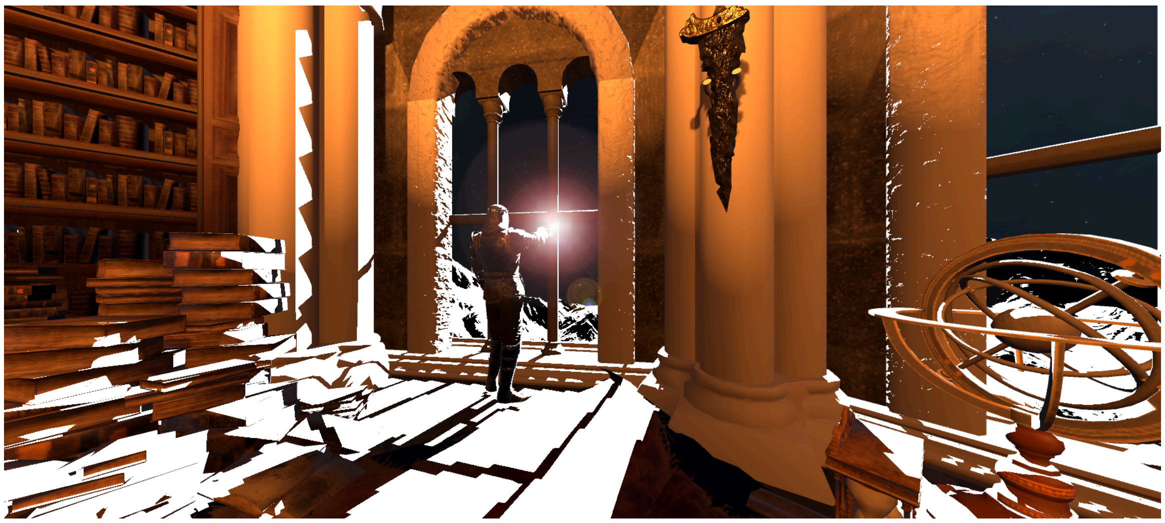


Figure 3. The Narrator casting the lightning flash

To get the user's full attention, the simulation begins with the narrator casting a lightning flash. First, the narrator switches from a Natural Idle animation to Pointing animation, and at the end of the animation, the lightning event is triggered. The lightning effect is achieved by quickly enabling and disabling a directional light with an intensity of 100 and a lens flare. There is a point light with a lens flare component at the position just in front of the narrator's fingertip. This point light is a child of the directional light, so it flickers with the directional light, making the narrator’s fingertip glow. The sound of thunder is played at the same time as the lightning. The purpose of the lightning is to attract the user's attention, to create mystery and curiosity about the story, and to signal to the user that the simulation has begun.

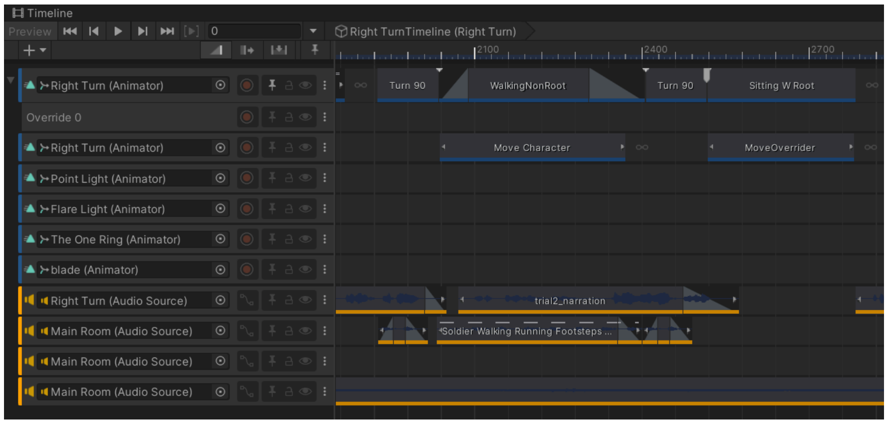


Figure 4. Timeline for Walking from Window to Chair and Sitting Animations

2.6.2 Walking from Window to Chair and Sitting

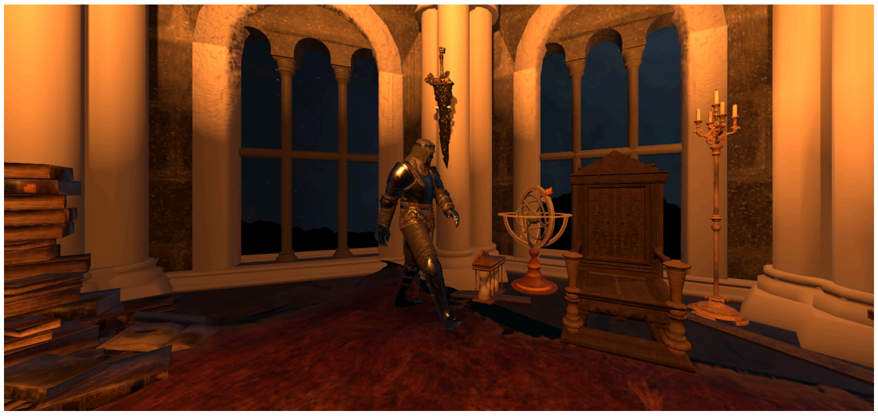


Figure 5. The Narrator walking towards the chair

From now on, the narrator begins to tell the story to the player. Meanwhile, he starts turning away from the window and walking towards the chair. This is achieved by combining the Right Turn 90 and Walking animations. There are also footstep sound effects in sync with the narrator's footsteps. When the narrator reaches the chair, another Right Turn 90 animation plays and then the narrator sits down.

2.6.3 The Narration

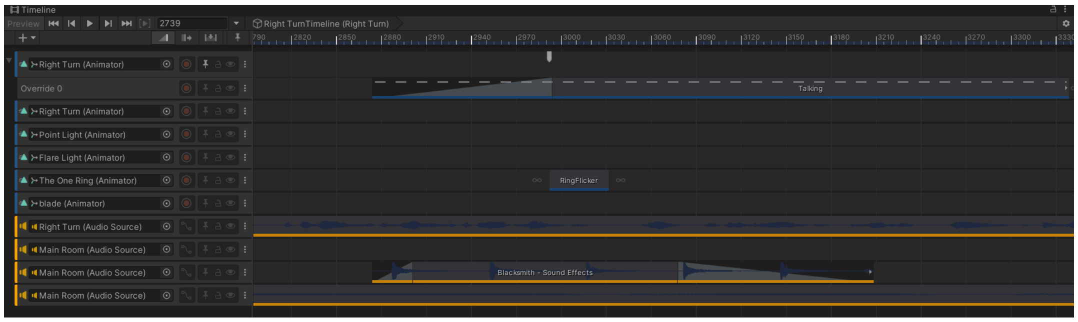


Figure 6. Timeline for story narration

When the narrator finishes sitting in the chair, he continues to tell the story. At this point, we start recording the user's gaze data into a CSV file, which will be explained in section **2.10**. The narrator remains in the chair for the rest of the simulation, alternating between different animations. We also play different sound effects throughout the simulation, such as ring forging sounds to army marching sounds. Both are done to keep the user's attention on the story. While the narrator is telling the story, several distractions take place to test the user's reaction time to find the likelihood of ADHD based on the study by Stokes et. al. [4].

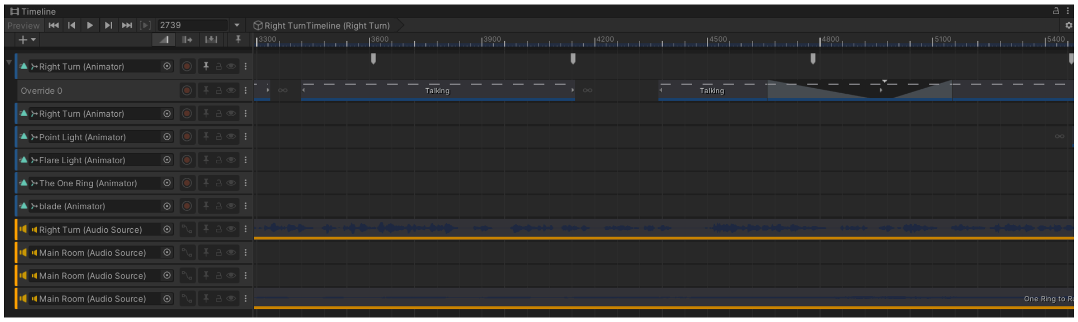


Figure 7. Timeline for story narration

2.6.4 Distractions

In total, we use 4 distractions in the simulation. The first distraction occurs when the narrator mentions the ring while the blacksmith sound effect is playing. In this distraction, the ring glows using the lens flare.

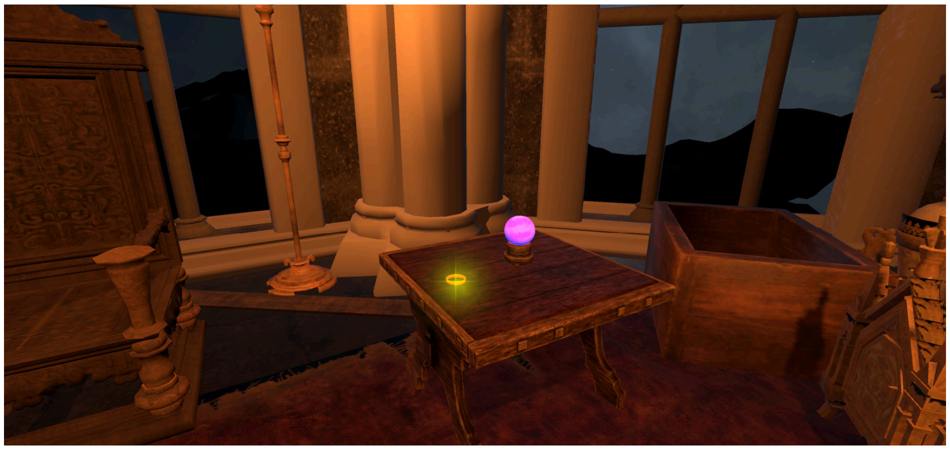


Figure 8. Ring Distraction

Another distraction used is the sword distraction. This distraction occurs while the narrator is talking about the rings gifted to the race of men, about 68 seconds into the simulation. The sword distraction is the only distraction that uses auditory stimulus. When the sword hits the ground, it plays the sword falling sound effect. This is done with the script PlaySoundOnHit.cs.

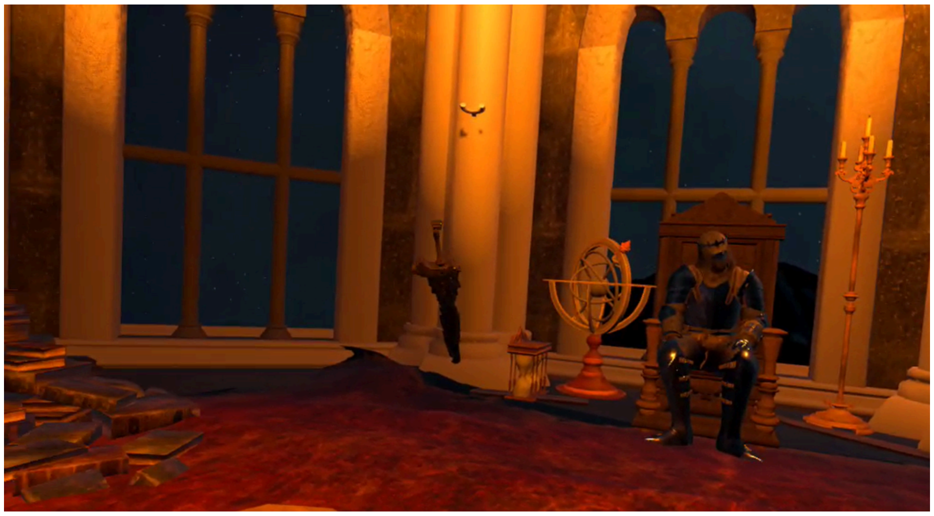


Figure 9. Sword Distraction

The final type of distraction is the Eye of Sauron distraction. Throughout the simulation, the Eye of Sauron was in the background. However, since there is no ambient light, it is not visible.

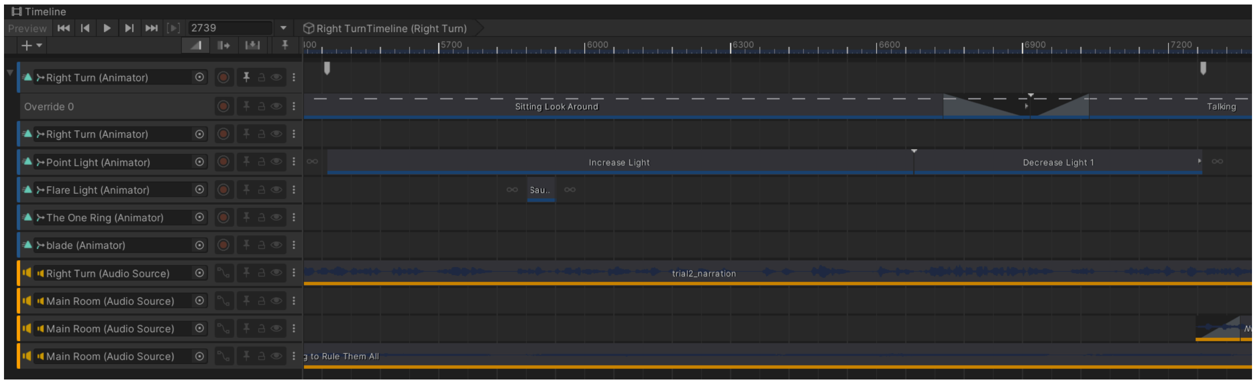


Figure 10. Timeline for Sauron Distraction

The Eye of Sauron distraction begins when the narrator mentions Dark Lord Sauron forging the master ring. First, the brightness of the point light shining on the Eye of Sauron is increased. About 7 seconds later, the Lens Flayer is introduced.

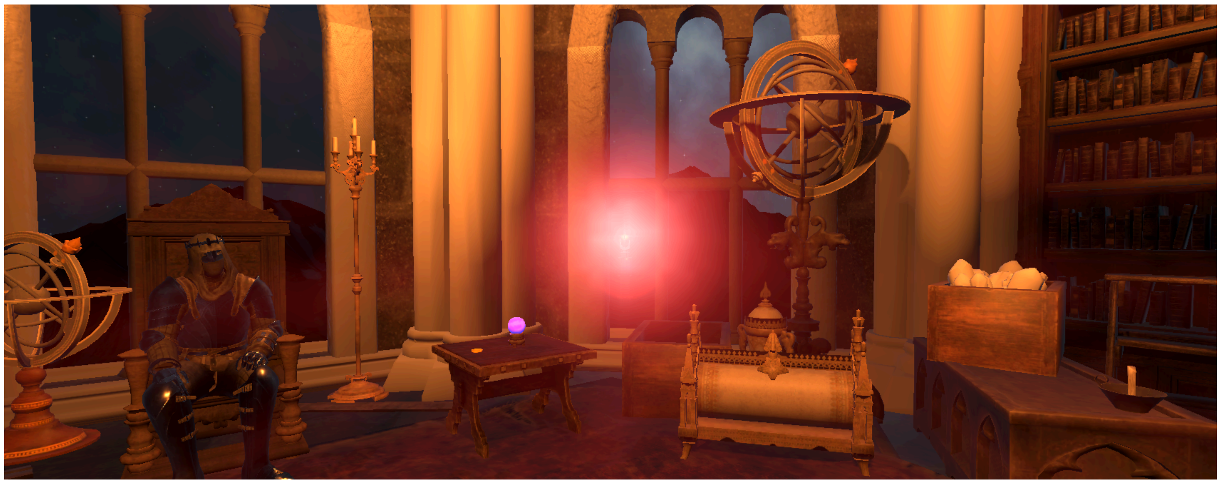


Figure 11. The Eye of Sauron Distraction

This lens flayer is the main distraction and is visible even when the Eye of Sauron is out of sight. This distraction is the longest distraction and gives the user more time to react to it.

2.6.6 Events

Each distraction has an Event game object to detect if the player is looking at that event. This game object is basically a sphere with a capsule collider and its mesh renderer disabled so that the user cannot see it in the simulation.

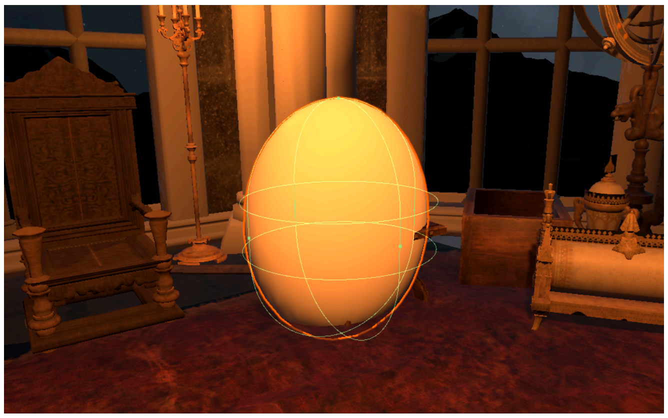


Figure 12. Event object for Ring Distraction

Attached to the CenterEyeAnchor is a cube object with a box collider and without a mesh renderer. When the user turns their head, this cube object turns with them. When this cube object collides with another collider, Unity calls the onTriggerEnter function (explained **2.10**) and we check if the colliding object is an 'Event' object. Each object has an Event.cs script that holds the name of the event and the time the event was triggered. By subtracting the current time from the time the event was triggered, we can find the amount of time it took for the user to react. Both the events and the cube attached to the camera have isTrigger enabled for their colliders. So that they go through other colliders and don't provide physical collisions.

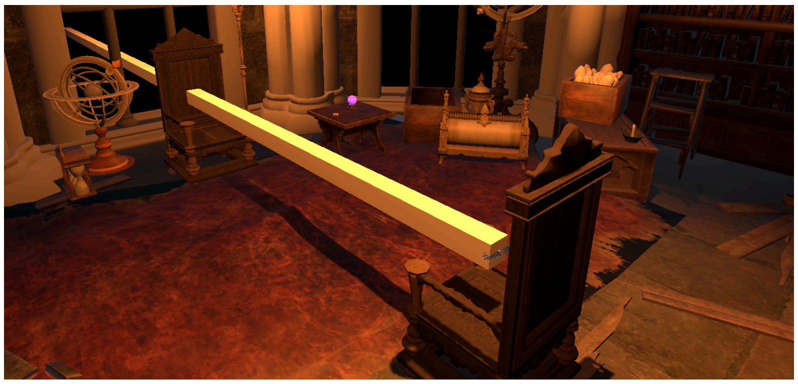


Figure 13. Box Collider Attached to the CenterEyeAnchor

2.6.7 Signals

To synchronize the animations with the distractions and events, we use signals. When a certain point in time is reached on the Timeline, it emits a certain signal. In Figure 14, signals are visible as white bookmark icons.

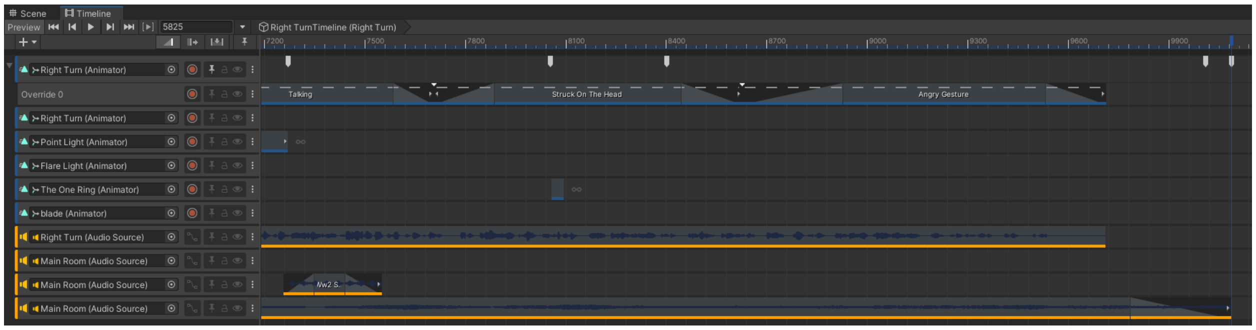
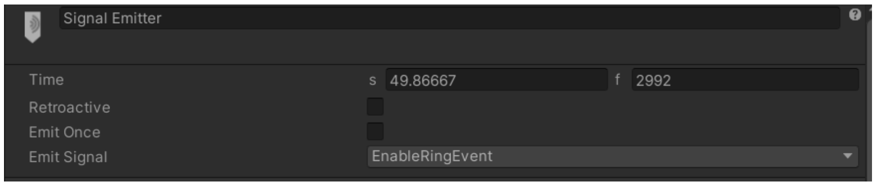


Figure 14. Timeline for End of Narration

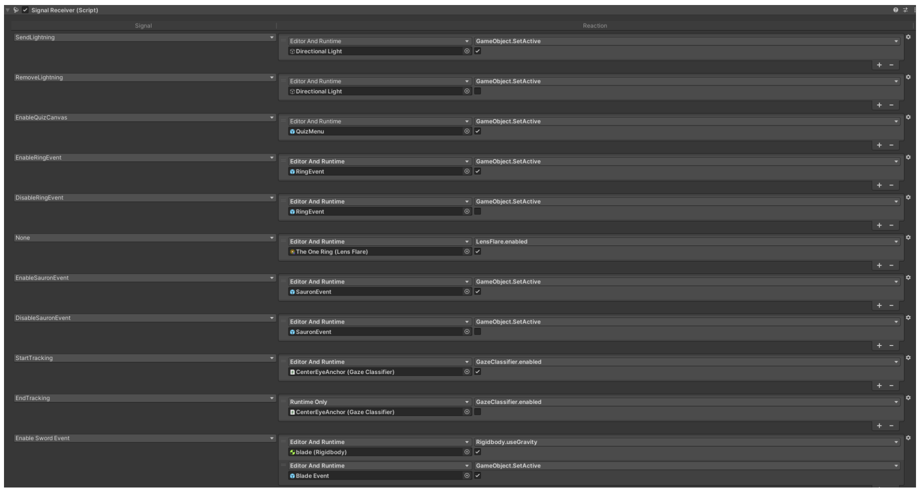


Signal Emitter

Time	s 49.86667 f 2992
Retroactive	☐
Emit Once	☐
Emit Signal	EnableRingEvent

Figure 15. Signal for Ring Event

The Narrator's game object has a Signal Receiver component that defines what each signal will do. When a signal is emitted from the timeline, Unity calls this component and performs the corresponding action.



Signal Receiver (Script)

Signal	Reaction			
SendLightning	Editor And Runtime	Directional Light	GameObject.SetActive	✓
RemoveLightning	Editor And Runtime	Directional Light	GameObject.SetActive	☐
EnableQuizCanvas	Editor And Runtime	QuizMenu	GameObject.SetActive	✓
EnableRingEvent	Editor And Runtime	RingEvent	GameObject.SetActive	✓
DisableRingEvent	Editor And Runtime	RingEvent	GameObject.SetActive	☐
None	Editor And Runtime	The One Ring (Lens Flare)	LensFlare.enabled	✓
EnableSauronEvent	Editor And Runtime	SauronEvent	GameObject.SetActive	✓
DisableSauronEvent	Editor And Runtime	SauronEvent	GameObject.SetActive	☐
StartTracking	Editor And Runtime	CenterEyeAnchor (Gaze Classifier)	GazeClassifier.enabled	✓
EndTracking	Runtime Only	CenterEyeAnchor (Gaze Classifier)	GazeClassifier.enabled	☐
Enable Sword Event	Editor And Runtime	blade (Rigidbody)	Rigidbody.useGravity	✓
	Editor And Runtime	Blade Event	GameObject.SetActive	✓

Figure 16. Signal Receiver Component

For example, for the Lightning event, we use the SendLightning and RemoveLightning signals. We send these signals one after the other in small time intervals to enable and disable the Directional Light game object. For distractions, the signals are used to enable the Event object for the corresponding distraction.

2.7 Quiz

After the narration ends, we stop recording the user's gaze and enable the Quiz canvas. The Quiz canvas is a ray-interactable menu having a text component and 4 buttons. We update the text and buttons on the Quiz canvas using the QA.cs script (explained later section **2.10**) and display whether the user’s answer is correct or incorrect.

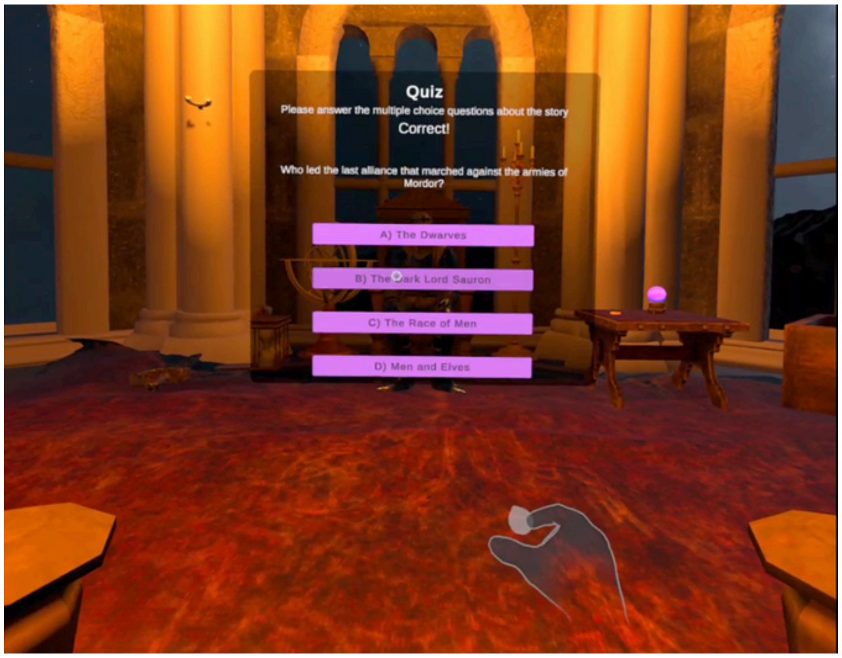

Quiz
Please answer the multiple choice questions about the story
Correct!
Who led the last alliance that marched against the armies of Mordor?

Figure 17. Quiz Canvas

2.8 Results Scene

When the quiz is finished, the simulation directs the user to the Results scene. In this scene, we have the Gaze Heatmap, the Minimum Response Time Graph, and the ADHD Probability data shown to the user. There are also Restart Simulation and Exit Simulation buttons.

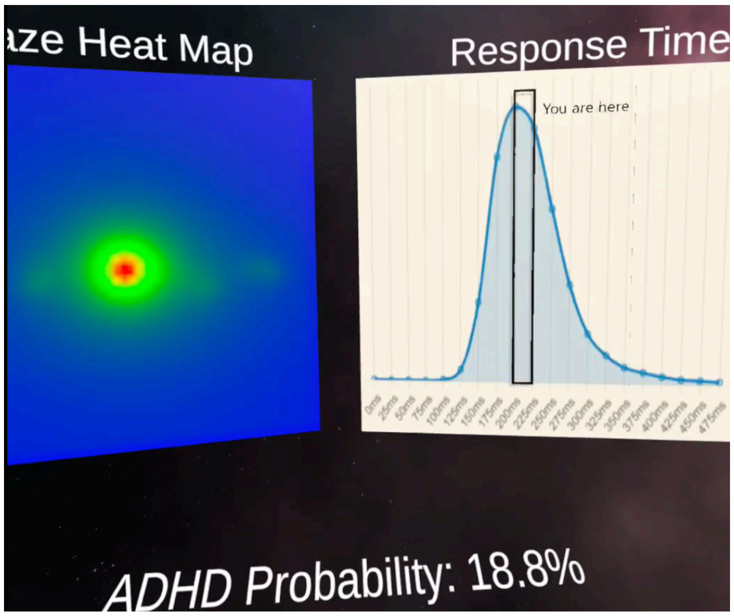
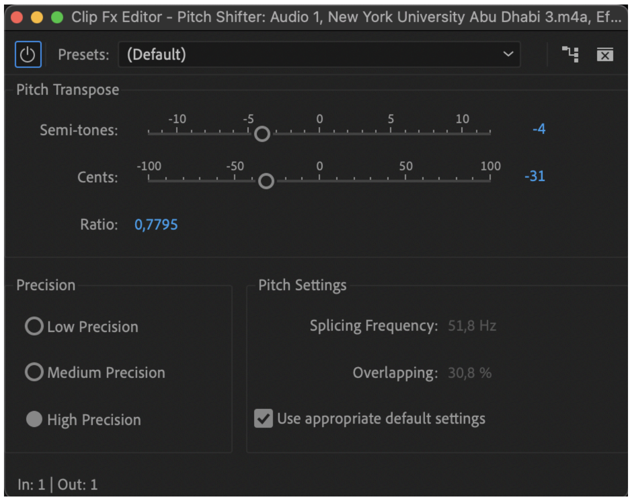


Figure 18. Results Scene

2.9 Narrator's Voice

Firstly, a clear narration with a decent audio recorder has been done. Then, Adobe Premiere Pro is utilized for voice effects. After an experimental sequence with many voice effects, the following steps are pursued: pitch and depth, modulation and distortion, ambiance and texture, mixing and finalization. Since Unity provides stereo and distant sound effects, this issue has been omitted from our consideration. For the pitch and depth, pitch shifter effect is used. Semi-tones level is chosen as -4 and cents level is chosen as -31, as this gives the best tone for our narrator. Parametric equalizer effect is also used while high pass and low pass filters help to decrease the spectrum of the sound to ensure the smoothness and the mysteriousness of the narrator voice at the same time and preserving the balance. Additionally, this prevents the unnatural "cracks" that happened after the previous (pitch shifter) effect.


Clip Fx Editor - Pitch Shifter: Audio 1, New York University Abu Dhabi 3.m4a, Ef...

Presets: (Default)

Pitch Transpose

Semi-tones: -4

Cents: -31

Ratio: 0,7795

Precision

○ Low Precision

○ Medium Precision

● High Precision

Pitch Settings

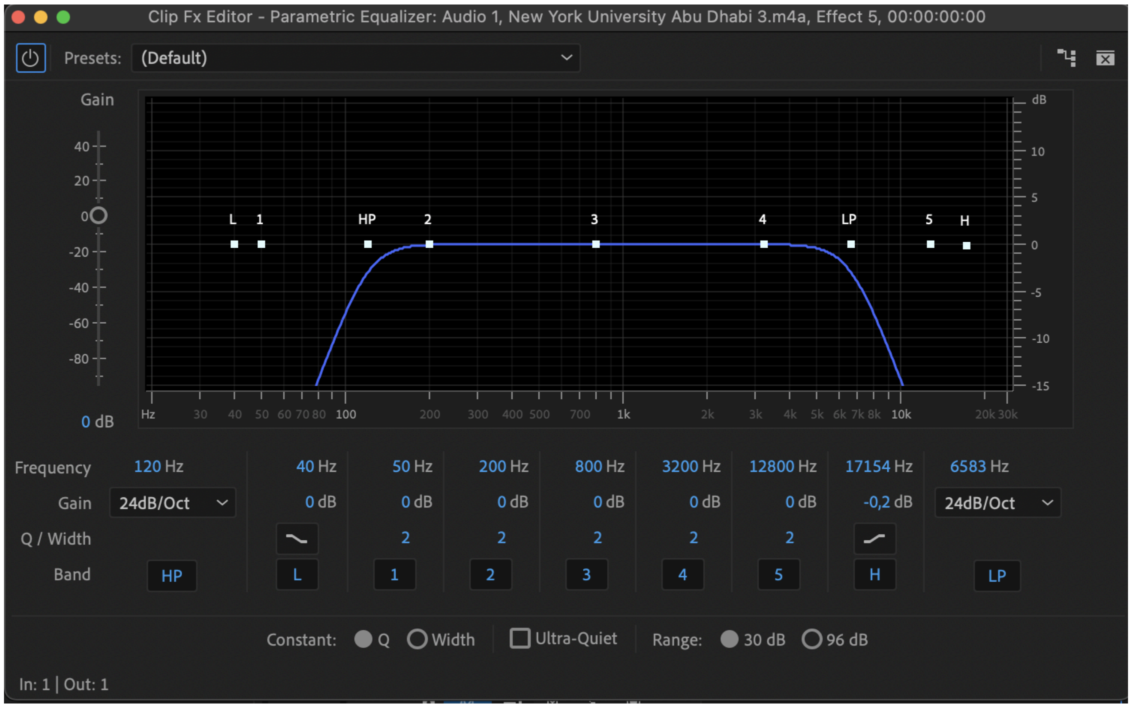
Splicing Frequency: 51,8 Hz

Overlapping: 30,8 %

✓ Use appropriate default settings

In: 1 | Out: 1

Figure 19. Pitch shifter parameters that are used to accommodate a certain tone to the narrator.



Clip Fx Editor - Parametric Equalizer: Audio 1, New York University Abu Dhabi 3.m4a, Effect 5, 00:00:00:00

Presets: (Default)

Gain: 0 dB

Frequency	120 Hz	40 Hz	50 Hz	200 Hz	800 Hz	3200 Hz	12800 Hz	17154 Hz	6583 Hz
Gain	24dB/Oct	0 dB	0 dB	0 dB	0 dB	0 dB	0 dB	-0,2 dB	24dB/Oct
Q / Width			2	2	2	2	2		
Band	HP	L	1	2	3	4	5	H	LP

Constant: ● Q ○ Width ☐ Ultra-Quiet Range: ● 30 dB ○ 96 dB

In: 1 | Out: 1

Figure 20. Parametric equalizer effect with high (HP) and low-pass (LP) filters utilized.

2.10 Data Processing and Code

Given below are the most significant C# Scripts that are used in the project, with some of their prominent functions and their description.

GazeClassifier.cs: Calculates and stores the angle at which the user's gaze is in the scene.

gazeAngles(): This function calculates the angle of the camera with respect to the z-axis. It first gets the forward direction of the camera. Then, it calculates the angles between the z-axis and the forward vector of the camera in the x and y directions. These angles are adjusted based on the scene and the sign of the gaze direction. The angles are then saved to a file using the saveGazeData() function.

saveGazeData(float x, float y): This function saves the gaze data to a file. It creates a string with the x and y values separated by a comma and a newline character. It then opens a StreamWriter to the file specified by gazefile, writes the gaze data string to the file, and closes the StreamWriter.

DisTimeClassifier.cs: Calculates and stores the response time of a user to visual stimuli.

`*OnTriggerEnter(Collider collision)*`: This is a Unity3D method that is called when the object this script is attached to enters a trigger collider. If the collided object has the tag "Event", it calculates the response time by subtracting the time the event was created (stored in the `Event` component of the collided object) from the current time. The collided object (the event) is then deactivated. Finally, it calls the `*saveResponseData(float resTime, string name)*` method to save the response time and event name to a file, similar to that in GazeClassifier.cs.

QA.cs: Implements the quiz; it loads and changes the quiz questions, and stores the user response.

`*parseQuestions()*`: This method is used to parse the questions from a hardcoded string array. Each line is split into its components and used to create a new Question object.

`*Question*` Class: This is a helper class that represents a single question in the quiz. It has a string for the question text, an array of strings for the options, an int for the correct answer (represented as an index into the options array), and a float for the difficulty of the question.

`*displayQuestion(Question q)*`: This method updates the UI to display the given question and its options.

`*answer(int answer)*`: This method is called when the player answers a question. It checks if the answer is correct, updates the result text, saves the answer data, and moves to the next question.

`*changeQuestion()*`: This method moves to the next question if there are any left, or loads the "Results" scene if there are no more questions.

DataProcessing.cs: parses, processes and calculates ADHD probability while displaying heatmap and response graph.

‘processData()’: Controls the execution of all calculations and display functions:

It first parses the gaze data using *parseGazeData()*, processes it with *processGaze(gazeData)*, and calculates statistics with *calcDataStats(classifications, "Gaze")*. It then creates a heatmap using *createHeatMap(gazeData)* and draws it with *drawHeatMap(heatmap)*. Similarly, it processes the Response and Quiz data. It also draws a reaction time panel with *drawReactionTime(responseData)*. It then calculates the user’s total score using *calcTotalScore(gazeStats, responseStats, quizStats)*. Finally, it displays the results using *displayResults(gazeStats, responseStats, quizStats, totalScore)*.

‘parseGazeData()’, parseResponseData(), ‘parseQuizData()’: These functions open the files for the gaze, response times and quiz answers, and load their data into an array which is subsequently returned.

‘processGazeData(float[][] data)’, ‘processResponseData()’, 'processQuizData()’: Iterates through each item in the respective array and calls the score calculation function. Stores the scores for each metric in an array and returns it.

‘getGazeScore(float x, float y)’: classifies the gaze point to a specific focus region (gazeBox), and returns the score associated with that box.

‘getResponseScore(float[] response)’: calculates the score depending on the response time for a specific visual stimuli based on the formula: $Response\ Score = \frac{1}{n} * \sum_{i=1}^{n} \sqrt{\frac{minEventTime_i}{userTime_i}}$, where n is the number of events, *minEventTime* is the minimum possible response time for a specific stimulus, and *userTime* is the actual response time of the user.

‘getAnswerScore(float correct, float diff)’: Calculates the score for a specific quiz result based on the answer and the difficulty of the question using the formula:
$QuizScore = \ correct * correctPts * diff$, where *correct* is a boolean showing if the user answer is correct, *correctPts* is the number of points allotted to a correct answer, *diff* is the difficulty of the question from 0 to 1.

‘calcDataStats(float[] data, string type)’: Calculates the average, standard deviation, sum of the scores and number of data points for a specific scores array, and returns these values.

‘calcTotalScore(float[] gazeStats, float[] responseStats, float[] quizStats)’: calculates the total score by summing the normalized scores for each metric and multiplying them by their weights.

`*createHeatMap(float[][] data)*`: This function creates a heatmap from the provided gaze data. It first calculates the maximum x and y values of the heatmap based on the maximum map angles and the distance to the plane. It then initializes the heatmap and iterates over the gaze data. For each gaze point, it checks if the point is within the heatmap boundaries. If it is, it calculates the x and y location of the gaze point on the heatmap, converts these to indices, and updates the heatmap values at these indices using `updateMapValues()`. Finally, it normalizes the heatmap using `normalizeMap()` and returns it.

`*updateMapValues(float[,] map, float x, float y)*`: This function updates the heatmap values around a given gaze point. It first calculates the bounds of the brush stroke based on the brush size. It then iterates over the heatmap values within these bounds and updates each value based on its distance from the gaze point. The closer the heatmap point is to the gaze point, the larger the value added to it.

`*drawHeatMap(float[,] map)*`: Draws the heat map onto a Unity texture. first creates a `Gradient` object, which is used to map heatmap values to colors. It then iterates over the heatmap values and adds the corresponding Color value into a 2D array. It converts the 2D color array to a 1D array and sets the pixels of the texture to these colors. It then applies this texture to a plane in the scene.

'*drawReactionTime(float[][] data)*': Displays the minimum reaction time on a normal distribution graph. It first calculates the minimum reaction time from the data. It then scales this time to fit the range of the graph. It calculates the position on the graph that this scaled time corresponds to. Finally, it moves the reaction showcase bar to this position to represent the statistical position of the user.

'*calcADHDProb(float score)*': Uses a predefined threshold to calculate the probability that the user does not have adhd. If the score is less than the threshold, the probability is calculated as half of the ratio of the score to the threshold. This will result in a probability between 0 and 0.5. If the score is equal to or greater than the threshold, the probability is calculated as 0.5 plus half of the ratio of the difference between the score and the threshold to the difference between 1 and the threshold. The probability that the user does have ADHD is calculated by subtracting the returned value from 1.

3. Results and Evaluation

The simulation was thoroughly tested across multiple test subjects, and the results were used to tune the parameters for data processing. In addition, we tested the following extreme cases:

Extreme Case	**Expected Result**
User looks at the narrator throughout the simulation.	Should achieve the max gaze score possible and heatmap should look red at the center point
User never looks at the narrator	Should achieve 0 gaze score and heatmap should be blue at the center
User looks at all the distractions the moment they get initiated	User should get the maximum possible responseTime score of 1
User never looks at the distractions	Should get minimum score of 0
User answers all quiz questions Correctly	Should achieve the quiz score of 1
User answers all quiz questions Incorrectly	Should achieve the quiz score of 0

Here are some of the results that were achieved after completing the simulation:

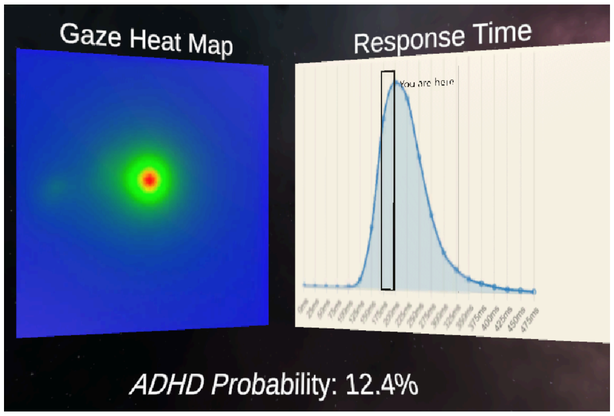


Fig. 21 Test with an average user

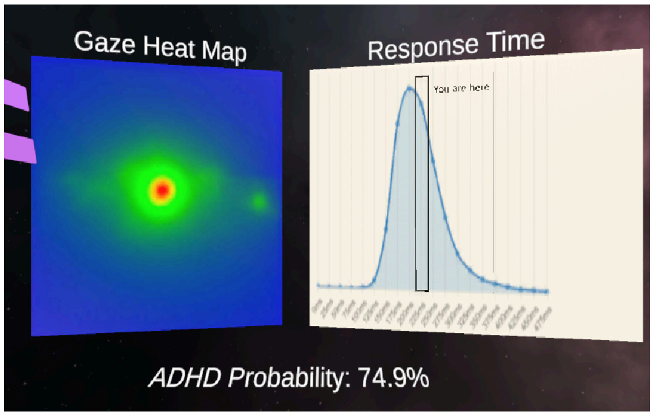
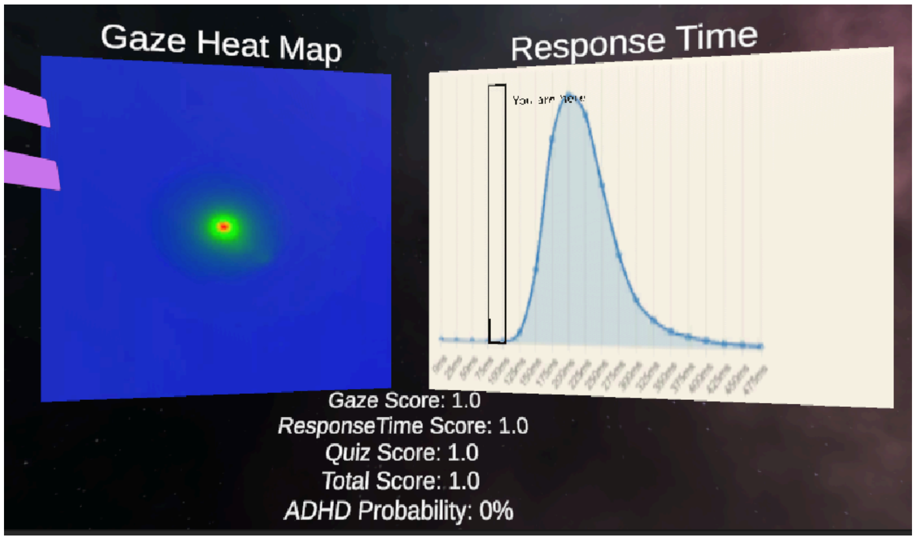


Fig. 22 Test with a user with ADHD



Gaze Score: 1.0
ResponseTime Score: 1.0
Quiz Score: 1.0
Total Score: 1.0
ADHD Probability: 0%

Fig. 23: Extreme Case Test with Maximum scores

Throughout the tests we have run we found out that lightning served as a great attention-getter and signal that the simulation has started. Also none of the user's faced problems with the user interface and found the simulation to be easy-to-use. The distractions were generally noticed by the users, however, some users that had not been paying attention to the narrator at the time missed the first Ring event, which served as a great way to measure user's attention to the narrator. One limitation we encountered with the simulation is that people sometimes reacted to distractions by moving their eyes, not their heads. Because we are unable to track their eye gaze, this caused the simulation to incorrectly assume that they missed some of the distractions.

3.2 Problems Encountered

- Lens Flare only working for one eye

The lens flare that we are using to alert players to events wasn't working properly on the headset for the first time. The flare was visible on the left eye, but not on the right. After searching the internet we couldn't find a proper solution until we tried adding post processing to the camera and it did the same thing as the lens flare, the left eye was fine but the right eye just showed white. While searching for a solution to this problem, we found that changing the Oculus rendering type from multipass to multiview solved both problems.

- Root motions from Mixamo animations did not apply to the rig

When we tried to sequence the animations from the Mixamo, the root motions on these animations did not apply to our 3D model. Basically, the rig's bone positions did not change with the animation, which caused the next animation to play from the bone's original position instead of the position from which the last animation left. We solved this problem with animation extrapolation, override layers and track offsets.

- Data file couldn't be read on Oculus which caused all code to crash

The first time we tried to save and retrieve data from a file in Oculus, our code crashed. Since we didn't have access to the debug console on Oculus, it took us some time to figure out what was going wrong. Then we realized that we couldn't write to the current application path in Oculus and instead had to use Application.permanentPath to read and write files.

- Raycasting not working on Quest 2

We originally planned to use raycasting to detect if the user is looking at an event. However, this didn't work on the VR headset, so instead we connected a rectangular box collider to the CenterEyeCameraAnchor, disabled the mesh renderer, and checked the collisions to see if the user was looking at the event.

- No eye gaze detection in Quest 2

Since we don't have access to eye gaze data on Quest 2, we instead increased the tolerance in our algorithm to account for the time difference between eye movement and head movement. We did this by changing our response score calculation function from $\frac{minEventTime}{userTime}$ to $\sqrt{\frac{minEventTime}{userTime}}$.

The reason we use the square root is that we have found that $\frac{1}{x}$ decreases very quickly and $\frac{1}{\sqrt{x}}$ scales much better. The graph in Figure 24 shows the difference between these two functions.

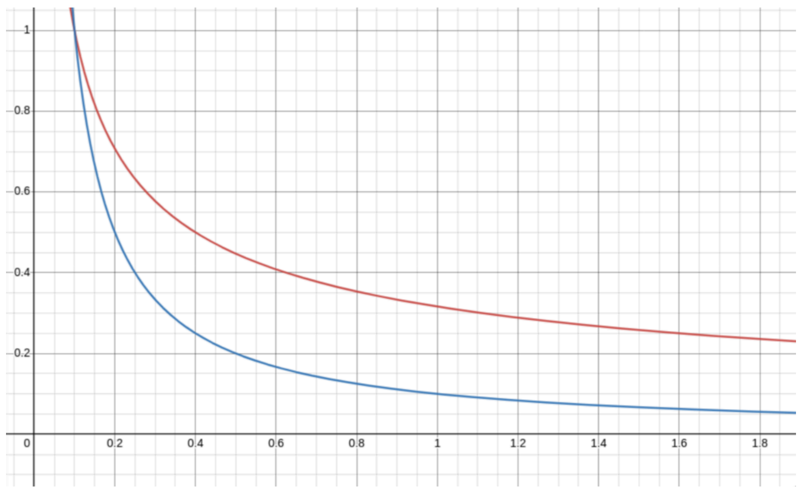


Figure 24. Comparison between $\frac{1}{x}$ (Blue) and $\frac{1}{\sqrt{x}}$ (Red) for minimum time value of 0.1

4. Conclusion and Future Work

This project serves as a framework for a virtual reality based ADHD diagnosis application. This software accelerates ADHD diagnosis, thereby reducing costs and increasing the accessibility of healthcare. The project assesses the user's attention throughout a simulation of a stimulus rich environment during a story narration and quiz. It uses three metrics for this: user gaze, stimulus response time, and quiz results. The data from all of these metrics are processed to compute the probability of a positive ADHD diagnosis with respect to predetermined thresholds. It also visualizes the given data through graphs and heatmaps, allowing a diagnosing psychiatrist to easily interpret and evaluate the user's condition. Though the software showcases fairly valid results, many further improvements can be made to increase its accuracy and improve user experience:

Eye Gaze Implementation: Due to the lack of eye tracking features in the Quest 2, the gaze tracking feature was not implementable in our project. Instead head tracking was implemented as a close alternative. In future versions, using headsets like the Quest Pro, the user's eye movements will be tracked, thereby giving a more accurate estimate of their attention and response times. This will lead to a more accurate assessment of the final result.

Further Testing: Due to time constraints, the project could not be tested rigorously with a large sample group. Being a biomedical tool, this application requires an immense amount of examination with both types of users - one's that do and do not have ADHD. This experimental data will allow for the fine-tuning of the data-processing parameters and thresholds, thereby allowing for a greater accuracy of the probability output.

Improved Graphics Implementation: The project currently uses SRP pipeline to render the scene, which displays the graphics at a lower quality than those made through the HDRP method. By using higher-end headsets and better texture quality, a more realistic scene can be made, thereby simulating a more lifelike environment for the user.

Story Variation: Due to the subjectivity of user preferences, they can be tested on a range of stories to get a better understanding of their overall attention level. Through this, biases regarding interest for/against the specific story are removed, resulting in a precise and impartial measurement of data.

5. Reflection on Learning

Throughout this project a diverse range of skills were developed ranging from logical programming to creative audio processing. The following list briefly describes the tools used and the skills developed during the creation of this software.

- C# Programming Language
    - File Management: Used in the storage and retrieval of user data
    - Data Processing: Learned to programmatically analyze raw data, and compute/extract the required information.
    - Heat Maps: Utilized 2D arrays to store and visualize data in an intuitive format
- Using Unity Game Engine for VR Games
    - Meta XR Framework
    - Timeline System: Used for sequencing the animations, events, and sounds
    - Animation System: Used for animating the narrator and events in the simulation
    - Collision and Trigger System: Used during the implementation of user response times to visual stimuli.
    - Scene and Texture System: Utilized during the visualization of processed data, especially the pixel-by-pixel presentation of the heatmap.
- Blender
    - 3D Model: Used for modifying 3D models for animating and creating objects to use in the scene
    - Rigging: Used for animating the narrator
- Adobe Premiere Pro
    - Audio Processing: Used extensively to record, edit and polish the story and corresponding sound effects.

This project enhanced our problem solving skills while increasing our knowledge regarding the Unity and Oculus frameworks. We learned to think algorithmically while implementing a modular way of ideation and development. It made us better communicators, while enhancing the research skills that were applied during the brainstorming and implementation process.

6. References

[1] What is ADHD?. Psychiatry.org - What is ADHD? (n.d.).

https://www.psychiatry.org/patients- families/adhd/what-is-adhd

[2] Diagnosis: attention deficit hyperactivity disorder (ADHD). NHS. (n.d.). NHS choices.

https://www.nhs.uk/conditions/attention-deficit-hyperactivity-disorder-adhd/diagnosis/

[3] Nigg JT, Butler KM, Huang-Pollock CL, Henderson JM. Inhibitory processes in adults with persistent childhood onset ADHD. J Consult Clin Psychol. 2002 Feb;70(1):153-7. doi: 10.1037//0022-006x.70.1.153. PMID: 11860041.

[4] Stokes, J. D., Rizzo, A., Geng, J. J., & Schweitzer, J. B. *Measuring attentional distraction in children with ADHD using virtual reality technology with eye-tracking*.Frontiers.

https://www.frontiersin.org/articles/10.3389/frvir.2022.855895/full

[5] Kleberg JL, Frick MA, Brocki KC. Eye-movement indices of arousal predict ADHD and comorbid externalizing symptoms over a 2-year period. Sci Rep. 23;13(1):4767. doi: 10.1038/s41598-023-31697-3. PMID: 36959373; PMCID: PMC10036637.

[6] Lee, D.Y., Shin, Y., Park, R.W. et al. Use of eye tracking to improve the identification of attention-deficit/hyperactivity disorder in children. Sci Rep 13, 14469 (2023).

https://doi.org/10.1038/s41598-023-41654-9