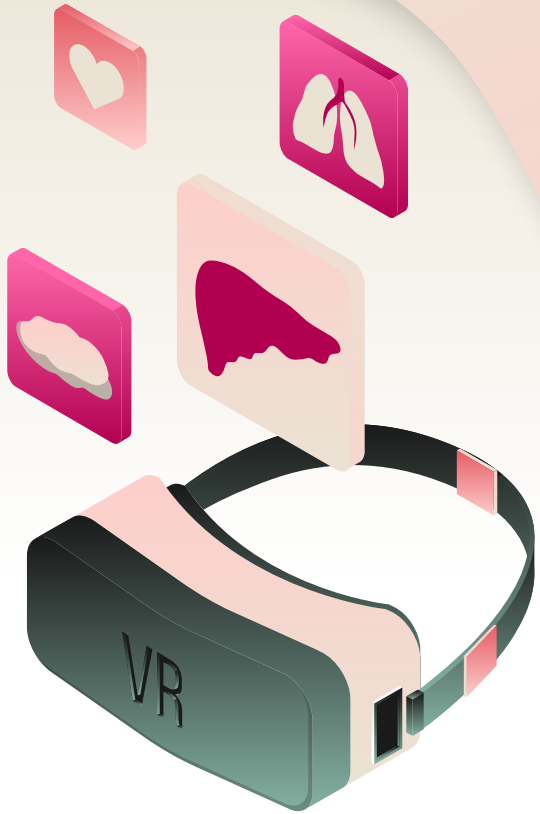
Using Virtual Reality to Teach Empathy to Nurses

Nursing Empathy VR Team:
Genevieve Lucas
Jaimie Mateer
Cierra Williams
Tori Broeker
Client: Lynsey Steinberg



OUTLINE

Project Overview

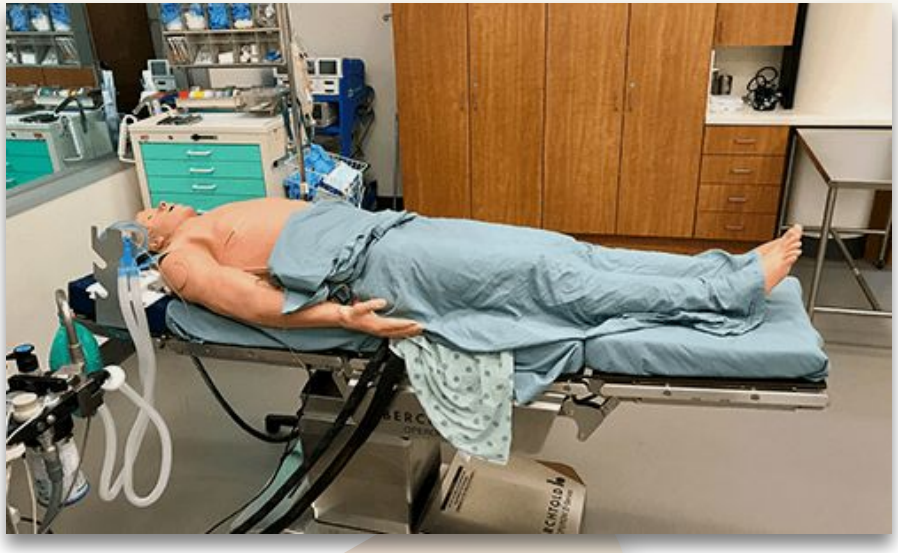
Can VR help hospice care nurses develop empathy for patients and their loved ones?

Empathy: the ability to understand and share the feelings of another

As a result of the pandemic, nurses have shown a tremendous need in developing empathy in palliative care nursing.

Existing Solutions

The Interdisciplinary Simulation Center is the primary healthcare simulation resource for the healthcare training programs of Augusta University.



Our Application

An immersive virtual reality training simulation for nursing students in order to improve cognitive awareness of empathetic understanding before having to enter the hospice care field.

VR Environment

Introduction Video

Training Simulation Videos

Sensor Data

Heartbeat

Pupillometry

Gaze Tracking

Feedback

Likert Scale Survey

Technology Used

Development Platform: Unity

- Version: 2021.3.12f1
- Primary language was C#

Head Mounted Display: HP Reverb G2 Headset

- PPG Sensor
- Tobii VR4 Integration Platform used to deliver accurate pupillometry and gaze-tracking data

APIS/SDKs

- Omnicept Core SDK
- Mixed Feature Reality Tool
- Tobii XR SDK
- OpenXR

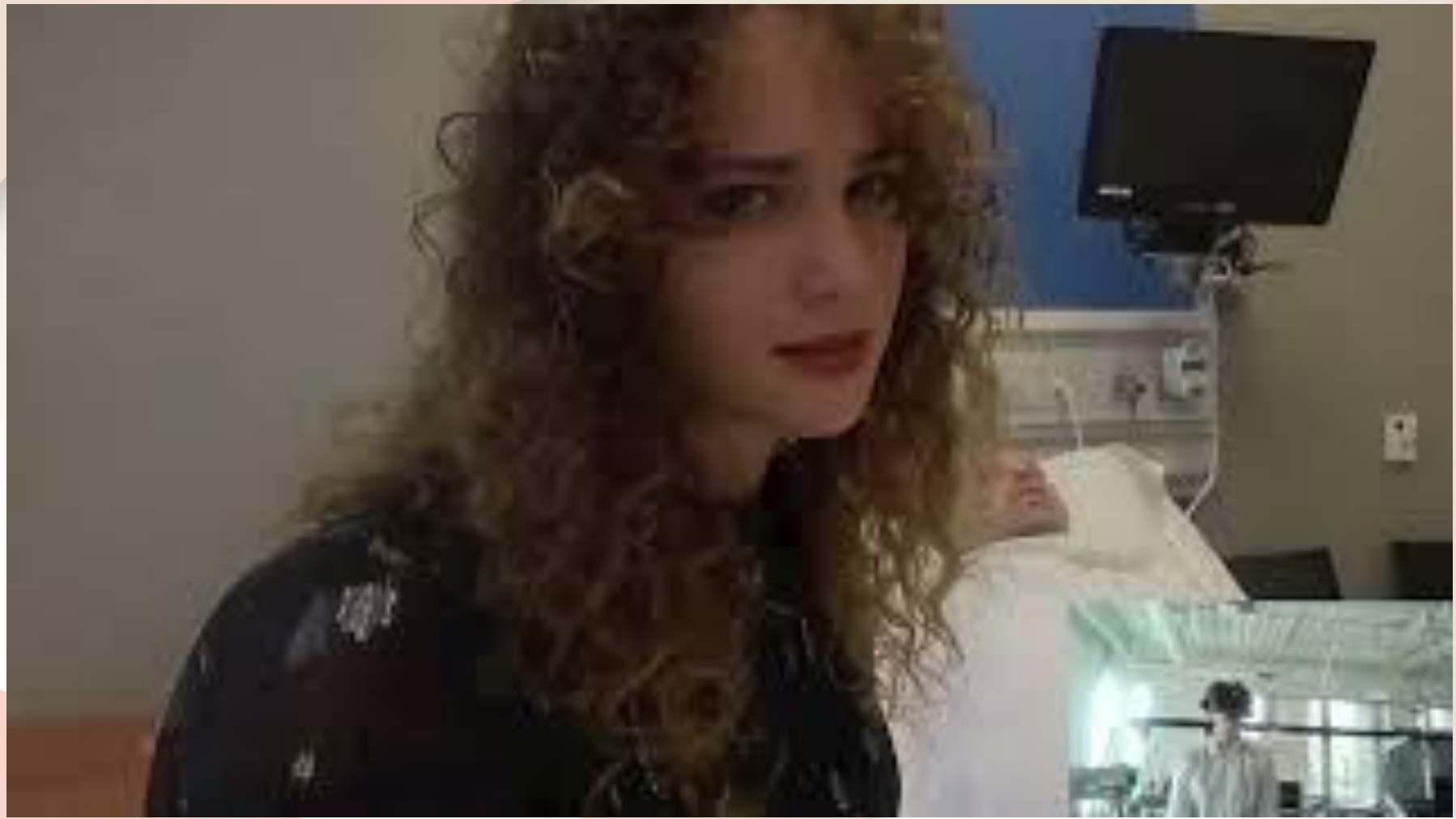
```
public void LoadRandomScene()
{


    if (!File.Exists(sceneFilePath) && !File.Exists(surveyFilePath))
    { // Load first random scene

        string rand = " ";
        StreamReader cr = new StreamReader(latinFilePath);
        for (int i = 0; i <= count % 12; i++)
        { rand = cr.ReadLine(); }
        string[] availableScenes = rand.Split(' ');
        int sceneNum = Int32.Parse(availableScenes[index]); // Obtain scene number
        index++;
        MakeVideoNameFile(sceneNum); // Write scene video file to temp file for data marking/labeling
        File.WriteAllLines(sceneFilePath, availableScenes); // Write remaining scene options to temp scene file
        SceneManager.LoadScene(sceneNum); // Load the first random scene

    }
    else if (!File.Exists(sceneFilePath) && File.Exists(surveyFilePath))
    { // Load goodbye scene

        while (!IsFileReady(surveyFilePath)) { } // Wait for survey file to be avalible before proceeding
        //while (!IsFileReady(sensorDataPath)) { } // Did not resolve issue

        List<string> surveyResponses = new List<string>(File.ReadAllLines(surveyFilePath)); // Read survey responses
        File.Delete(surveyFilePath); // Delete temp survey file

        TextWriter tw = new StreamWriter(sensorDataPath, true);

        tw.WriteLine(); // Added for spacing
        tw.WriteLine(); // Added for spacing
        tw.WriteLine(); // Added for spacing

        for (int i = 0; i < surveyResponses.Count; i++)
        {
            tw.WriteLine($"Likert Response #{i + 1}    " + surveyResponses[i]);
        }

        tw.WriteLine(); // Added for spacing
        tw.WriteLine(); // Added for spacing
        tw.WriteLine(); // Added for spacing

        tw.Close();
```

```
        SceneManager.LoadScene(7);// loading goodbye scene
        // !!!  KEEP IN MIND THIS  ^^^  WOULD NEED TO UPDATED EVERYTIME YOU WANTED TO ADD A SCENE TO THE RANDOM LIST  !!!
    }
    else
    { // This is not the first scene...
        List<string> availableScenes = new List<string>(File.ReadAllLines(sceneFilePath)); // Load file contents in list

        if (index == 3)
        { // Load last random scene
            File.Delete(sceneFilePath); // Delete temp file
            MakeVideoNameFile(Int32.Parse(availableScenes[index])); // Write scene video file to temp file for data marking/labeling
            SceneManager.LoadScene(Int32.Parse(availableScenes[index])); // Load last scene
        }
        else
        { // Load a random scene

            int sceneNum = Int32.Parse(availableScenes[index]); // Obtain scene number
            index++;
            MakeVideoNameFile(sceneNum); // Write scene video file to temp file for data marking/labeling
            File.WriteAllLines(sceneFilePath, availableScenes); // Update scene file
            SceneManager.LoadScene(sceneNum); // Load the random scene
        }
    }
}
```

```
public class DataWrapper : MonoBehaviour
{
    1 reference
    Pupil pupilObj = new Pupil();
    1 reference
    GazeTracking gazeObj = new GazeTracking();
    1 reference
    HeartBPM heartObj = new HeartBPM();
    11 references
    TextWriter tw;
    1 reference
    int frameNum = 1;
    2 references
    private string sensorDataPath = @"..\SensorData.csv";

    // Start is called before the first frame update
    0 references
    void Start()
    {
        //Creating sensor data file, writing User ID, and writing header fields for sensor data
        tw = new StreamWriter(sensorDataPath,false); // false means write-mode (will overwite)

        // Writing User ID (Ideally this would be moved to the intro video scene)
        tw.WriteLine("User ID");
        tw.WriteLine($"{new System.Random().Next()}");
        tw.WriteLine(); // Added for spacing
        tw.WriteLine(); // Added for spacing
        tw.WriteLine(); // Added for spacing

        // Writing header fields for sensor data then closing file
        tw.WriteLine("Time-Stamp, Left Pupil Dilation, Right Pupil Dilation, Avg. Pupil Dilation, Left Gaze Tracking, Right Gaze Tracking, Combined Gaze, Heart Rate");
        tw.Close();
    }

    // Update is called once per frame
    0 references
    void Update()
    {
        //if(frameNum % 30 == 0){ // writes to CSV every 30 frames (can be changed)
        tw = new StreamWriter(sensorDataPath,true); // true means append-mode (will NOT overwite)
        tw.WriteLine(System.DateTime.Now.ToString("yyyy.MM.dd.hh.mm.ss.ffffff") + "," + pupilObj.GetData() + "," + gazeObj.GetData() + "," + heartObj.GetData());
        tw.Close();
        //}

        frameNum++;
    }

}
```

```
using System.Threading.Tasks;
using System.IO;
using HP.Omnicept.Unity; //this connects to the gliaBehaviour script

2 references
public class Pupil : MonoBehaviour
{
    3 references
    private GliaBehaviour _gliaBehaviour = null;
    7 references
    private float leftPupilDilation = 0;
    7 references
    private float rightPupilDilation = 0;
    6 references
    private float baselineLeftPupilDilation = 0;
    6 references
    private float baselineRightPupilDilation = 0;
    4 references
    private float baselineAveragePupilDilation = 0;
    4 references
    private bool hasTakenBaseline = false;
    3 references
    string filePath = @"C:\Users\ARVR Lab\Documents\pupil.txt"; // For testing in lab only
    3 references
    private GliaBehaviour gliaBehaviour //this defines gliaBehaviour
    {
        get
        {
            if (_gliaBehaviour == null)
            {
                _gliaBehaviour = FindObjectOfType<GliaBehaviour>();
            }

            return _gliaBehaviour;
        }
    }
    // Start is called before the first frame update
    0 references
    void Start()
    {
        if (hasTakenBaseline != true)
        {
            var pupillometry = gliaBehaviour.GetLastEyeTracking();
            //set the baseline values and mark that the baseline has been taken
            baselineLeftPupilDilation = pupillometry.LeftEye.PupilDilation;
            baselineRightPupilDilation = pupillometry.RightEye.PupilDilation;
            baselineAveragePupilDilation = (baselineLeftPupilDilation + baselineRightPupilDilation) / 2;
            hasTakenBaseline = true;

            //write the baseline values to a file
            //using (var writer = new StreamWriter("pupil_dilation.txt", true))

            using (StreamWriter writer = new StreamWriter(filePath, true))
            {
                writer.WriteLine("Baseline Left Pupil Dilation: " + baselineLeftPupilDilation);
                writer.WriteLine("Baseline Right Pupil Dilation: " + baselineRightPupilDilation);
                writer.WriteLine("Baseline Average Pupil Dilation: " + baselineAveragePupilDilation);
            }
        }
    }
```

```
    void Update()
    {
        var pupillometry = gliaBehaviour.GetLastEyeTracking();
        if (pupillometry != null)
        {
            //get the left and right pupil dilation values
            leftPupilDilation = pupillometry.LeftEye.PupilDilation;
            rightPupilDilation = pupillometry.RightEye.PupilDilation;

            if (hasTakenBaseline != true)
            {
                //set the baseline values and mark that the baseline has been taken
                baselineLeftPupilDilation = leftPupilDilation;
                baselineRightPupilDilation = rightPupilDilation;
                baselineAveragePupilDilation = (baselineLeftPupilDilation + baselineRightPupilDilation) / 2;
                hasTakenBaseline = true;

                //write the baseline values to a file
                //using (var writer = new StreamWriter("pupil_dilation.txt", true))

                using (StreamWriter writer = new StreamWriter(filePath, true))
                {
                    writer.WriteLine("Baseline Left Pupil Dilation: " + baselineLeftPupilDilation);
                    writer.WriteLine("Baseline Right Pupil Dilation: " + baselineRightPupilDilation);
                    writer.WriteLine("Baseline Average Pupil Dilation: " + baselineAveragePupilDilation);
                }
            }
            //calculate the average pupil dilation
            var averagePupilDilation = (leftPupilDilation + rightPupilDilation) / 2;

            using (StreamWriter writer = new StreamWriter(filePath, true))
            {
                writer.WriteLine("Left Pupil Dilation: " + leftPupilDilation);
                writer.WriteLine("Right Pupil Dilation: " + rightPupilDilation);
                writer.WriteLine("Average Pupil Dilation: " + averagePupilDilation);
            }
        }
    }


    1 reference
    public string GetData(){
        var pupillometry = gliaBehaviour.GetLastEyeTracking();
        if (pupillometry != null)
        {
            //get the left and right pupil dilation values
            leftPupilDilation = pupillometry.LeftEye.PupilDilation;
            rightPupilDilation = pupillometry.RightEye.PupilDilation;
            //calculate the average pupil dilation
            var averagePupilDilation = (leftPupilDilation + rightPupilDilation) / 2;

            string pupilData = leftPupilDilation.ToString() + "," + rightPupilDilation.ToString() + "," + averagePupilDilation.ToString();
            return pupilData;
        }
        else{
            return "PD 1 null,PD 2 null,PD 3 null";
        }
    }
```



Issues / Challenges

Team hardware limitations

Omnicept did not work well with Steam VR

Omnicept headset has not been used in the AR/VR lab

Discrepancies in HP documentation

Many APIs to use

Future Work

Add interactibility to the intro video and training simulation videos

Fix video audio issues

Face colliders for how often the tester is looking at the face of a loved one

Speech-To-Text function to record the verbal response of testers

QUESTIONS?