RV Educational Institutions ®
RV College of Engineering ®
Autonomous Institution Affiliated to Visvesvaraya Technological University, Belagavi
Approved by AICTE, New Delhi

Go, change the world®

VIRTUAL REALITY

AND

AUGMENTED REALITY

Experiential Learning
on
AR Puzzler App
(18IS72)

Submitted by

Abdu Rehman
(1RV20IS001)

Aryan Wani
(1RV20IS012)

2023-24

Introduction

This report introduces an innovative Augmented Reality (AR) application tailored for puzzle enthusiasts - AR Puzzler. This application aims to revolutionize the puzzle-solving experience by seamlessly integrating digital and physical dimensions, offering users an immersive journey into the world of enigmatic challenges. Augmented Reality technology, renowned for its ability to merge virtual elements with reality, empowers users to interact with puzzles in a three-dimensional space, transcending conventional methods and elevating the entire puzzle-solving process.

In the dynamic landscape of AR Puzzler, users embark on captivating adventures where puzzles come to life in unprecedented ways. Leveraging AR technology, the application provides users with hands-on exploration of intricate puzzle mechanics, transforming every solve into an immersive and engaging experience. Through dynamic visualizations and interactive elements, complex puzzle concepts are simplified, enabling users to delve deeper into the mysteries of each challenge.

Moreover, in educational settings, AR Puzzler serves as a valuable tool to cultivate a deeper understanding of problem-solving strategies and spatial reasoning skills. Whether one is a novice seeking to hone puzzle-solving prowess or a seasoned enthusiast craving new challenges, AR Puzzler promises to revolutionize the way individuals engage with puzzles. Through this comprehensive overview, the significance of AR Puzzler and its potential to redefine puzzle-solving dynamics, fostering creativity, and cognitive development are underscored.

Tools/Assets used

1. Unity

Unity stands as the cornerstone of AR Puzzler, offering a powerful and adaptable game development framework. Its cross-platform capabilities and vast repository of assets render it an optimal choice for crafting immersive augmented reality puzzles. By harnessing Unity's capabilities, we seamlessly fuse diverse elements, guaranteeing a seamless and interactive user interface for navigating through our puzzle challenges.

2. ARKit

For iOS devices, ARKit served a similar purpose to ARCore, allowing the app to create immersive augmented reality experiences tailored for Apple's mobile ecosystem.

3. Lean Touch

In order to optimize touch interactions within the AR Puzzle Explorer app, the development team chose to integrate Lean Touch, a dedicated asset tailored for gesture recognition and touch input management within Unity. This asset was selected for its ability to streamline the implementation of touch controls, ultimately providing users with an intuitive and highly responsive interface. By leveraging Lean Touch, the app's usability has been significantly enhanced, allowing users to navigate through puzzles and interact with elements seamlessly.

Execution Steps

1.Project Initialization and ARCore Integration:

The execution begins with initializing a new project tailored for augmented reality (AR) within Unity. ARKit is seamlessly integrated, laying the foundation for the app's AR functionalities. This platform enables the app to understand the real-world environment, facilitating features such as environmental understanding and plane detection crucial for creating immersive AR experiences.

2.Importation of 3D Models and Audio Assets:

Detailed 3D models of puzzle components are imported into the project as objects, ensuring accuracy and realism in representing various puzzle elements. These models are created in-house by providing accurate coordinates, rotation and scaling , depending on the requirements.

3.Lean Touch Integration:

Lean Touch, a Unity asset designed for gesture recognition and touch input handling, is integrated into the project to optimize touch interactions within the AR application. This step enables users to intuitively interact with puzzle components through gestures such as tapping, dragging, and scaling, enhancing the overall user experience. Lean Drag Translate and Lean Twist Rotate Axis components are used.

4.Canvas Creation and Button Integration:

A user interface (UI) is created within the Unity environment to provide users with intuitive controls and navigation options. A horizontally scrollable menu is created using ScrollView which contains the pieces of puzzles as buttons. Once a button is clicked, only that particular object or piece of puzzle can be moved.

5.Lean Drag Translate ,Lean Twist Rotate Axis, and Rescaling Scripts:

Individual scripts are developed to handle interactions with puzzle components, enabling functionalities such as dragging, rotating, and snapping pieces into place. These scripts ensure that users can manipulate puzzle elements seamlessly within the AR environment, providing a realistic and immersive puzzle-solving experience.

6. AR Raycasting for Table Placement:

Individual scripts which contain the exact dimensions for approximate positioning and placement of objects in the puzzle are developed to handle interactions with puzzle components, enabling functionalities such as dragging, rotating, and snapping pieces into place. These scripts ensure that users can manipulate puzzle elements seamlessly within the AR environment, providing a realistic and immersive puzzle-solving experience.

7. Method Creation for Object Manipulation:

In AR Puzzle Explorer, we developed methods for moving, rotating, and scaling puzzle pieces. This modular approach facilitated efficient coding and management, providing precise control over the behavior of various puzzle components within the augmented reality environment.

8. Coordinate-Based Method for Fixing CPU Parts:

To enhance user engagement a coordinate-based method was implemented to facilitate the precise placement of puzzle pieces within the augmented reality environment. By utilizing predetermined coordinates, this method streamlined the positioning process, enabling users to accurately align and examine puzzle components seamlessly.

9. Testing and Optimization:

The app undergoes rigorous testing to identify and address any bugs or performance issues. Optimization techniques are implemented to ensure smooth performance and compatibility across various devices.

10. Scene Importation and Mobile App Build:

In the final phase of development for the application , the scene was imported and the augmented reality application was built for deployment on mobile devices. This critical step ensured the smooth delivery of the AR experience to users, enabling them to embark on an immersive and engaging journey into the world of AR puzzle-solving directly from their mobile devices.

Code snippet

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ButtonScipts : MonoBehaviour
{
    private Vector3[] positions;
    [SerializeField] private GameObject parent;
    [SerializeField] private GameObject p1;
    [SerializeField] private GameObject p2;
    [SerializeField] private GameObject p3;
    [SerializeField] private GameObject p4;
    private int touchCountp1 = 0, touchCountp2 = 0, touchCountp3 = 0, touchCountp4 = 0;
    private List<GameObject> objects = new List<GameObject>();
    // Start is called before the first frame update
    void Start()
    {
        positions = new Vector3[4];
        positions[0] = new Vector3(0.00057f, 0, -0.026f);
        positions[1] = new Vector3(-0.035f, 0, 0.06f);
        objects.Add(p1);
        objects.Add(p2);
        objects.Add(p3);
        objects.Add(p4);

        p1.transform.localPosition = new Vector3(-0.2f, 0.15f, 0.25f);
        p2.transform.localPosition = new Vector3(-0.1f, 0.15f, 0.25f);
        p3.transform.localPosition = new Vector3(0.1f, 0.15f, 0.25f);
        p4.transform.localPosition = new Vector3(0.2f, 0.15f, 0.25f);

        foreach (GameObject item in objects)
        {
            item.GetComponent<Lean.Touch.LeanDragTranslate>().enabled = false;
            item.GetComponent<Lean.Touch.LeanTwistRotateAxis>().enabled = false;
        }
    }
```

Figure 1: ButtonScipts Class: Manages the behavior of buttons and associated objects

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class DynamicScrollView : MonoBehaviour
{
    [SerializeField] private Transform scrollViewContent;
    [SerializeField] private GameObject prefab;
    [SerializeField] private List<Sprite> puzzles;

    private void Start() {
        foreach (Sprite item in puzzles) {
            GameObject newItem = Instantiate(prefab, scrollViewContent);
            if(newItem.TryGetComponent<ScrollViewItem>(out ScrollViewItem scrollViewItem)) {
                scrollViewItem.changeImage(item);
            }
        }
    }
}
```

Figure 2: DynamicScrollView Class: Manages the dynamic content of a scroll view

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.EventSystems;

public class ScrollViewItem : MonoBehaviour, IPointerClickHandler
{
    [SerializeField] private Image child;

    public void changeImage (Sprite image) {
        child.sprite = image;
    }

    public void OnPointerClick (PointerEventData eventData) {
        Debug.Log("Clicked item");
    }
}
```

Figure 3: ScrollViewItem Class: Represents an item within a scroll view and handles click events

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class SnapPositions : MonoBehaviour
{
    [SerializeField] private GameObject parent;
    [SerializeField] private GameObject p1;
    [SerializeField] private GameObject p2;
    [SerializeField] private float xPos;
    [SerializeField] private float zPos;
    private Vector3[] positions;
    void Start()
    {
        positions = new Vector3[4];
        positions[0] = new Vector3(0.00057f, 0.15f, -0.026f);
        positions[1] = new Vector3(-0.035f, 0.15f, 0.06f);
    }
    void Update()
    {
        // Check if parent component is visible in AR
        if (parent.GetComponent<Renderer>().isVisible)
        {
            // Debug.Log("Parent position: " + parent.transform.position);
        }
        // if (p1.GetComponent<Renderer>().isVisible)
        // {
        //     // p1.transform.position = parent.transform.position + positions[0];
        // }
        // if (p2.GetComponent<Renderer>().isVisible)
        // {
        //     // p2.transform.position = parent.transform.position + positions[1];
        // }
        float d1 = (p1.transform.localPosition - parent.transform.localPosition).magnitude;
        if(d1 <= 0.06 && d1 >= 0.025){
            p1.transform.position = parent.transform.position + positions[0];
        }
    }
}
```

Figure 4: SnapPositions Class: Snaps child objects to predefined positions relative to a parent object