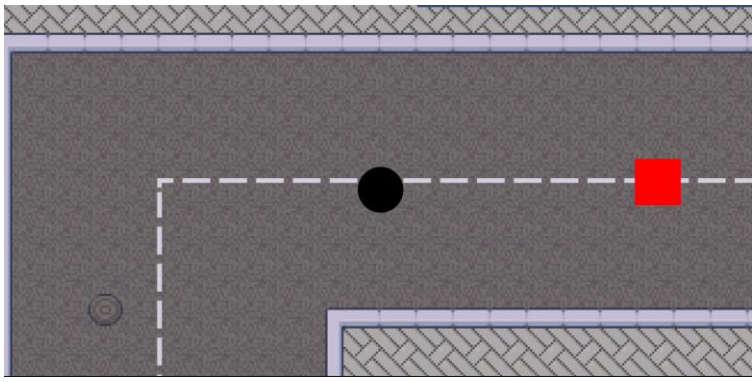
Unused Features and Work Outcomes

Author: LIYUANFAN 2020003927

1. GreyBox



This part was completed before game development began, so it was not uploaded to GitHub. Here is a playable link.

(Unity WebGL Player | VRProject_2DGreyBox)

The server is privately owned and will be available until early July.

In game development, the GreyBox represents the direction of the game, showcasing the general gameplay and actual effects. The purpose of this Grey Box is to help the development team better understand what game they are developing.

Overview: The Grey Box developed for the initial bus theme assumes that the black circle represents the bus, and the red square is the mission location. It demonstrates the approximate effect the game will have upon completion. The bus itself has an inertia system, making it difficult to control. Touching the red square will trigger an NPC dialogue about the history and culture of Ansan. The

dialogue system includes a complete workflow, where the dialogue changes based on options, and choosing all the correct options leads to a special storyline.

Both the Task system and the dialogue system are demonstrated here. This version of the dialogue system did not used in public game. It is a complete workflow based on Ink and Inky. Due to its specialization, it was not provided to team members. A simplified version of the system was later developed for the team.

Code (only a portion is shown due to its large volume, please experience the actual effect by playing the game):

Inky (Dialogue System) Workflow:

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

0 Unity 脚本 | 0 个引用
public class QuestionNpc : MonoBehaviour
{
    public GameObject Player;

    public static event Action<Story> OnCreateStory;

    0 Unity 消息 | 0 个引用
    void Awake()
    {
        // Remove the default message
        RemoveChildren();
        //StartStory();
    }

    0 Unity 消息 | 0 个引用
    private void OnTriggerEnter2D(Collider2D collision)
    {
        StartStory();
        Player.GetComponent<Rigidbody2D>().velocity = Vector2.zero;
    }

    // Creates a new Story object with the compiled story which we can then play!
    1 个引用
    void StartStory()
    {
        story = new Story(inkJSONAsset.text);
        if (OnCreateStory != null) OnCreateStory(story);
        RefreshView();
    }

    // This is the main function called every time the story changes. It does a few things:
    // Destroys all the old content and choices.
    // Continues over all the lines of text, then displays all the choices. If there are no choices, the story is finished!
    2 个引用
    void RefreshView()
    {
        // Remove all the UI on screen
        RemoveChildren();

        // Read all the content until we can't continue any more
        while (story.canContinue)
        {
            // Continue gets the next line of the story
            string text = story.Continue();
            // This removes any white space from the text.
            text = text.Trim();
            // Display the text on screen!
            CreateContentView(text);
        }

        // Display all the choices, if there are any!
        if (story.currentChoices.Count > 0)
        {
            for (int i = 0; i < story.currentChoices.Count; i++)
            {
                Choice choice = story.currentChoices[i];
                Button button = CreateChoiceView(choice.text.Trim());
                // Tell the button what to do when we press it
                button.onClick.AddListener(delegate {
                    OnClickChoiceButton(choice);
                });
            }
        }
        // If we've read all the content and there's no choices, the story is finished!
        else
        {
            Button choice = CreateChoiceView("End of story.\nRestart?");
            choice.onClick.AddListener(delegate {
                //StartStory();
```

```
    // When we click the choice button, tell the story to choose that choice!
    1 个引用
    void OnClickChoiceButton(Choice choice)
    {
        story.ChooseChoiceIndex(choice.index);
        RefreshView();
    }

    // Creates a textbox showing the the line of text
    1 个引用
    void CreateContentView(string text)
    {
        Text storyText = Instantiate(textPrefab) as Text;
        storyText.text = text;
        storyText.transform.SetParent(canvas.transform, false);
    }

    // Creates a button showing the choice text
    2 个引用
    Button CreateChoiceView(string text)
    {
        // Creates the button from a prefab
        Button choice = Instantiate(buttonPrefab) as Button;
        choice.transform.SetParent(canvas.transform, false);

        // Gets the text from the button prefab
        Text choiceText = choice.GetComponentInChildren<Text>();
        choiceText.text = text;

        // Make the button expand to fit the text
        HorizontalLayoutGroup layoutGroup = choice.GetComponent<HorizontalLayoutGroup>();
        layoutGroup.childForceExpandHeight = false;

        return choice;
    }

    // Destroys all the children of this gameobject (all the UI)
    3 个引用
    void RemoveChildren()
    {
        int childCount = canvas.transform.childCount;
        for (int i = childCount - 1; i >= 0; --i)
        {
            Destroy(canvas.transform.GetChild(i).gameObject);
        }
    }

    [SerializeField]
    private TextAsset inkJSONAsset = null;
    public Story story;

    [SerializeField]
    private Canvas canvas = null;

    // UI Prefabs
    [SerializeField]
    private Text textPrefab = null;
    [SerializeField]
    private Button buttonPrefab = null;
}
```

BUS Movement System:

```
    Unity 脚本(1 个资产引用)|0 个引用
public class MoveController : MonoBehaviour
{
    public float speed = 5f;
    public float deceleration = 0.5f;

    private Vector2 velocity;

    Unity 消息|0 个引用
    void Update()
    {
        // 获取键盘输入
        float horizontalInput = Input.GetAxis("Horizontal");
        float verticalInput = Input.GetAxis("Vertical");

        // 计算速度增量
        Vector2 inputVector = new Vector2(horizontalInput, verticalInput);
        velocity += inputVector * speed * Time.deltaTime;

        // 应用惯性减速
        velocity *= Mathf.Clamp01(1f - deceleration * Time.deltaTime);

        // 移动物体
        transform.Translate(velocity * Time.deltaTime);
    }
}
```

:

2. Photo-taking Feature

Grab ball will take a photo

Finally fixed Git ,this project only can use Github to push , git is unable to access this project, very strange .

This branch was used to test the photo-taking functionality, but it is no longer needed for planned changes outside of the photo-taking functionality, so it has been deprecated. This branch contains some test scenes and cameras (balls).

About the photo function:

Picking up the ball in the scene will automatically take a picture, and the picture will be saved in the running memory, which can be found through the log information address. The output system has not been produced yet.

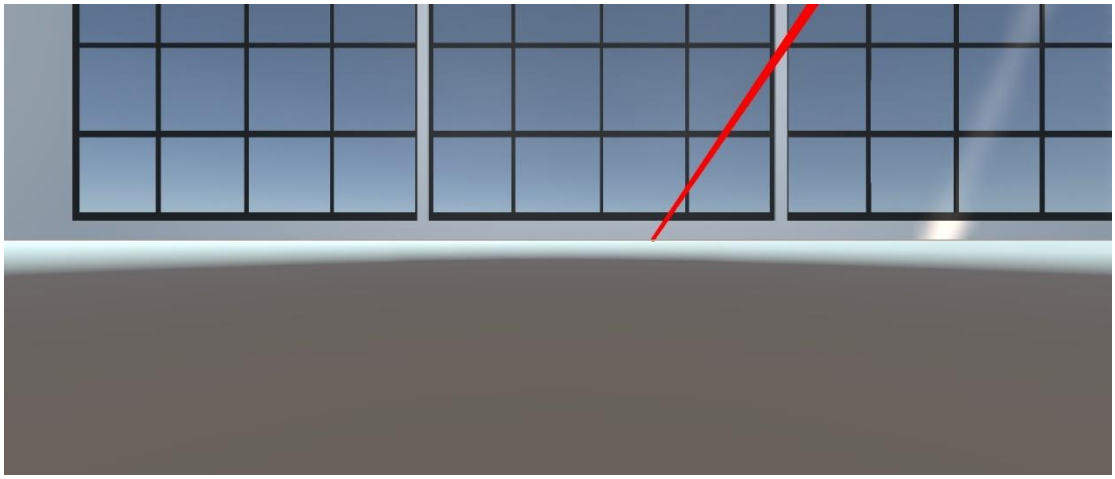
Stage_3

Originally designed for the Stage 3 reporter role-playing game project that I was responsible for, but it was abandoned due to a change in direction. It is stored in the Stage3 branch on GitHub.

The current code takes an automatic photo when an object is grabbed (the photo is stored in the cache).

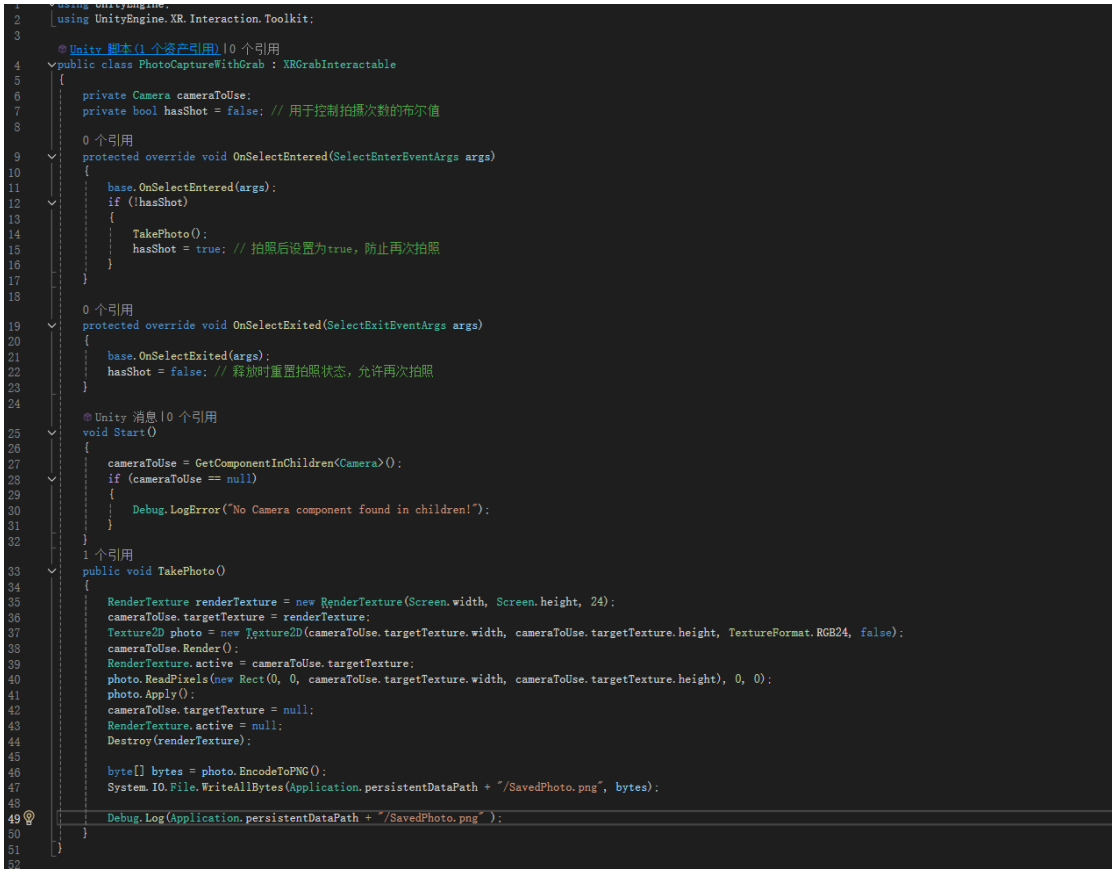
In-Game:

Photo Taken from the Sphere Camera Angle:



This includes a complete framework for grabbing objects and interaction.

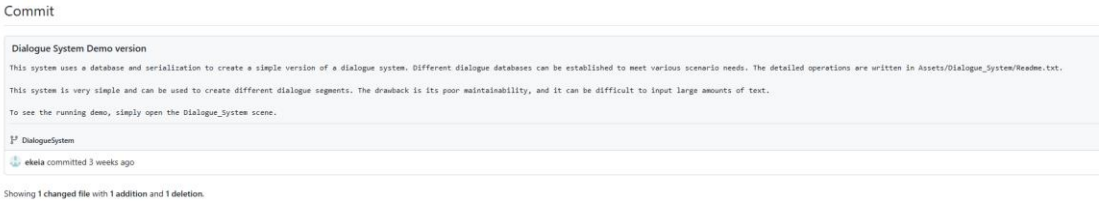
Below is a part of the code:



```
using UnityEngine;
using UnityEngine.XR.Interaction.Toolkit;

Unity 脚本(1 个资产引用)|0 个引用
public class PhotoCaptureWithGrab : XRGrabInteractable
{
    private Camera cameraToUse;
    private bool hasShot = false; // 用于控制拍摄次数的布尔值

    0 个引用
    protected override void OnSelectEntered(SelectEnterEventArgs args)
    {
        base.OnSelectEntered(args);
        if (!hasShot)
        {
            TakePhoto();
            hasShot = true; // 拍照后设置为true，防止再次拍照
        }
    }

    0 个引用
    protected override void OnSelectExited(SelectExitEventArgs args)
    {
        base.OnSelectExited(args);
        hasShot = false; // 释放时重置拍照状态，允许再次拍照
    }

    Unity 消息|0 个引用
    void Start()
    {
        cameraToUse = GetComponentInChildren<Camera>();
        if (cameraToUse == null)
        {
            Debug.LogError("No Camera component found in children!");
        }
    }
    1 个引用
    public void TakePhoto()
    {
        RenderTexture renderTexture = new RenderTexture(Screen.width, Screen.height, 24);
        cameraToUse.targetTexture = renderTexture;
        Texture2D photo = new Texture2D(cameraToUse.targetTexture.width, cameraToUse.targetTexture.height, TextureFormat.RGB24, false);
        cameraToUse.Render();
        RenderTexture.active = cameraToUse.targetTexture;
        photo.ReadPixels(new Rect(0, 0, cameraToUse.targetTexture.width, cameraToUse.targetTexture.height), 0, 0);
        photo.Apply();
        cameraToUse.targetTexture = null;
        RenderTexture.active = null;
        Destroy(renderTexture);

        byte[] bytes = photo.EncodeToPNG();
        System.IO.File.WriteAllBytes(Application.persistentDataPath + "/SavedPhoto.png", bytes);

        Debug.Log(Application.persistentDataPath + "/SavedPhoto.png");
    }
}
```

Because it was abandoned, the subsequent parts were not completed. The original plan was to record the objects in the center of the camera when taking photos, and combine this with the task system to create a mini-game where players complete tasks by taking continuous photos.

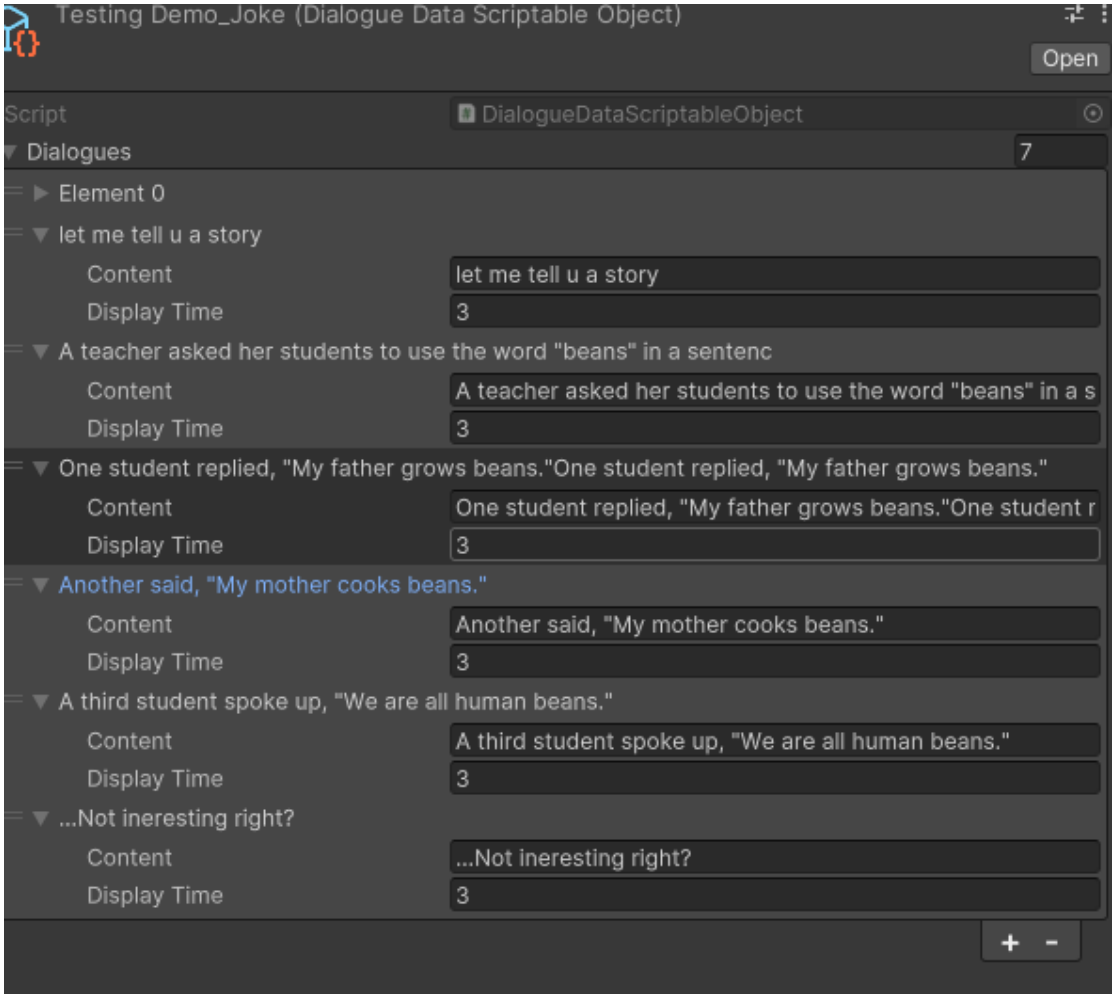
3. Dialogue System



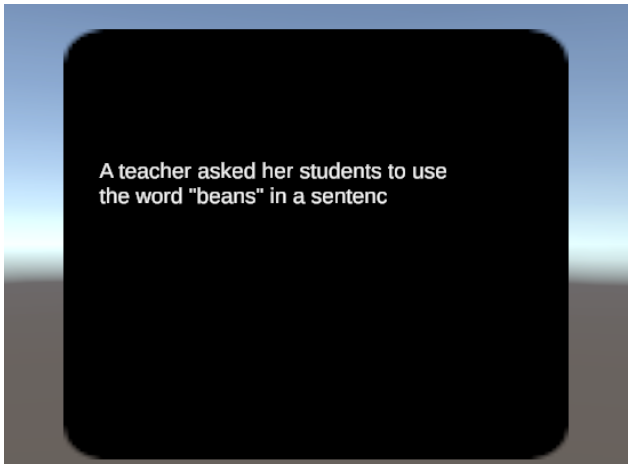
Creating a dialogue system itself is not difficult, but this system was designed to be more manageable and controllable for developers.

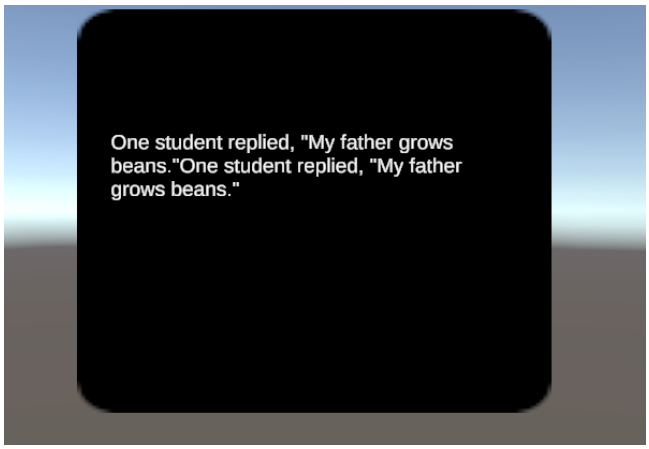
In the Grey Box, I also developed a dialogue system based on Ink architecture with a complete workflow. Due to concerns about its usability threshold, a simplified version was redeveloped. It is stored in the Dialogue_System branch on GitHub.

Inspector Panel:

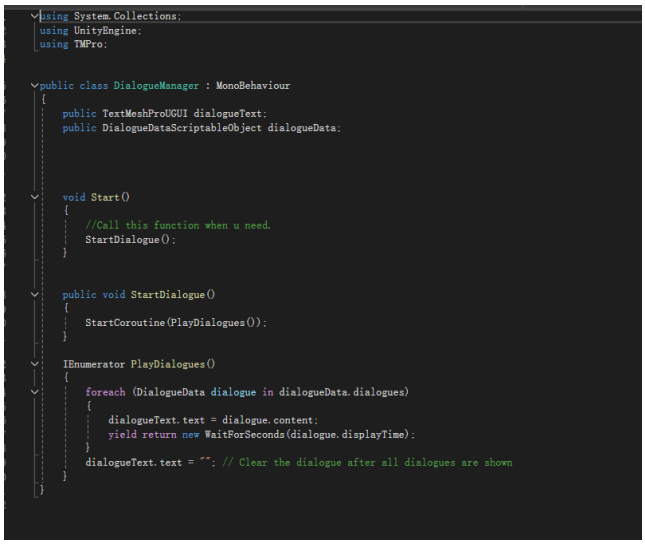


In Game:

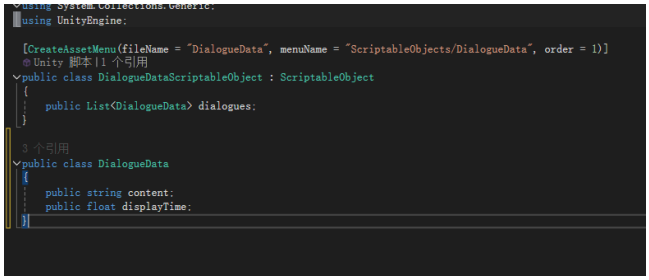




Code Part:



```
using System.Collections;
using UnityEngine;
using TMPro;


public class DialogueManager : MonoBehaviour
{
    public TextMeshProUGUI dialogueText;
    public DialogueDataScriptableObject dialogueData;




    void Start()
    {
        //Call this function when u need.
        StartDialogue();
    }


    public void StartDialogue()
    {
        StartCoroutine(PlayDialogues());
    }

    IEnumerator PlayDialogues()
    {
        foreach (DialogueData dialogue in dialogueData.dialogues)
        {
            dialogueText.text = dialogue.content;
            yield return new WaitForSeconds(dialogue.displayTime);
        }
        dialogueText.text = ""; // Clear the dialogue after all dialogues are shown
    }
}
```



```
using UnityEngine;

[CreateAssetMenu(fileName = "DialogueData", menuName = "ScriptableObjects/DialogueData", order = 1)]
Unity 脚本 | 1 个引用
public class DialogueDataScriptableObject : ScriptableObject
{
    public List<DialogueData> dialogues;
}

3 个引用
public class DialogueData
{
    public string content;
    public float displayTime;
}
```

Overview: Using a database and Unity's serialization features, I created a dialogue database system that can be modified and added to within the inspector window. Each dialogue can control the playback time, length, and font size to help developers achieve the desired performance effect.