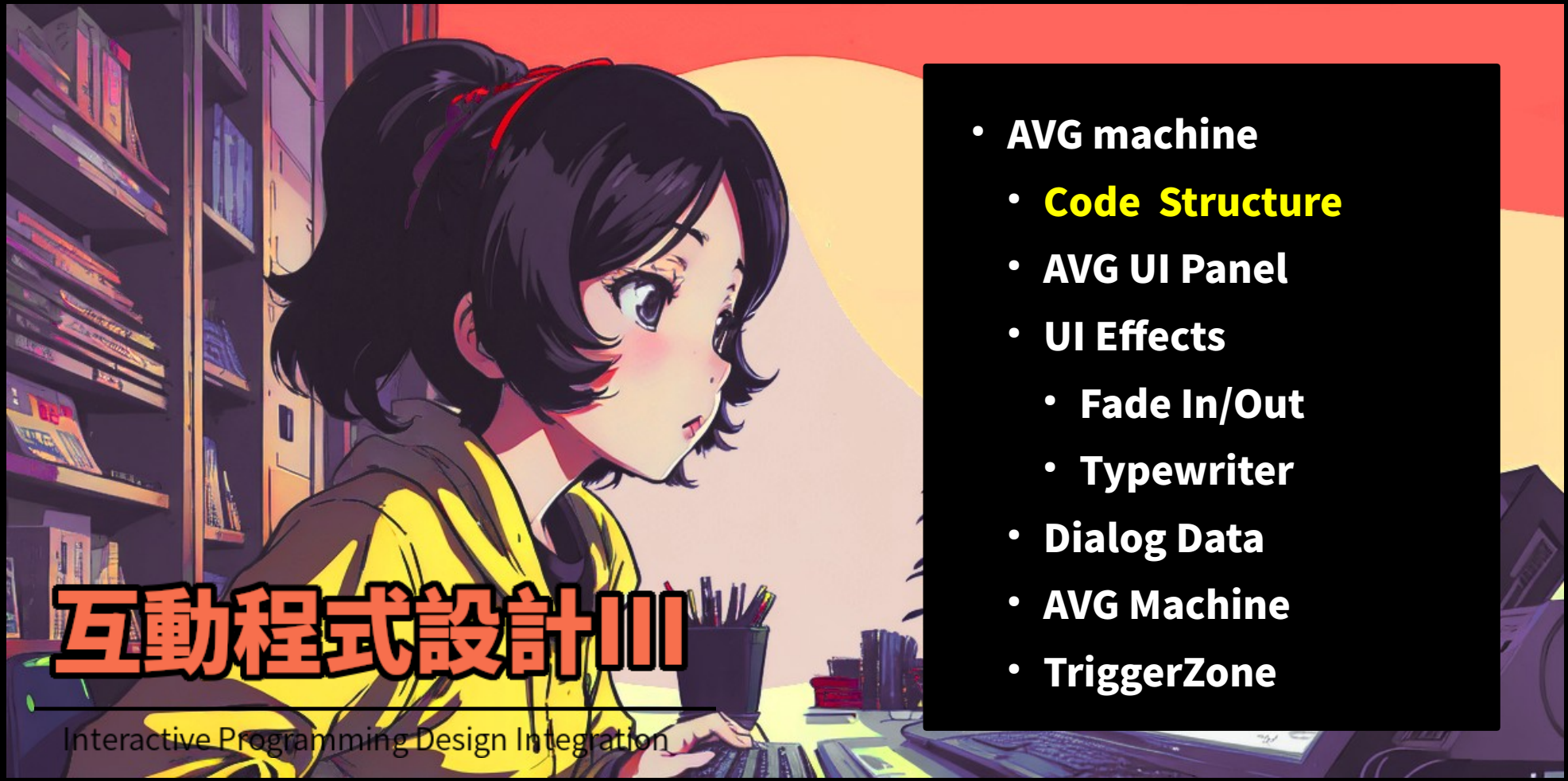

互動程式設計III
Interactive Programming Design Integration



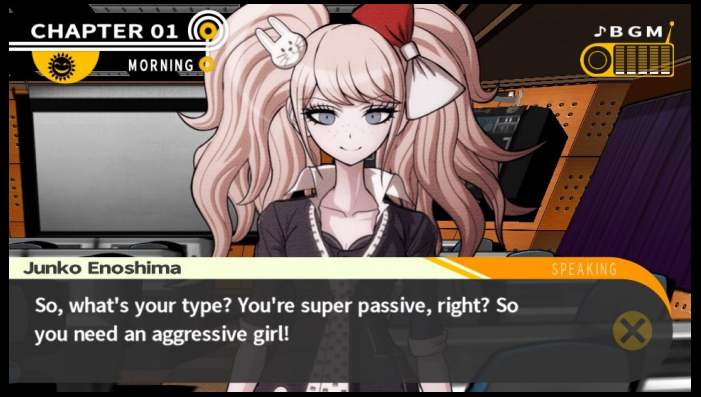
- AVG machine
  - Code Structure
  - AVG UI Panel
  - UI Effects
    - Fade In/Out
    - Typewriter
  - Dialog Data
  - AVG Machine
  - TriggerZone

Adventure Game (AVG) & Visual Novel (VN)

- Read the introduction of Adventure Game(AVG) on wikipedia [*]. Put a little bit more on Visual Novel(VN) parts. This is the type we are going to implement in this lesson.
  - Adventure Game (AVG):A broad category encompassing various sub-genres with unique gameplay styles and features.
  - Visual Novel (VN):A specialized sub-genre of AVG, known for its distinct screen layouts and narrative-driven flow.VN focuses heavily on storytelling, often featuring branching dialogues and player choices.
- Notable VN Titles:Explore examples of remarkable VN games here:
  - https://geekireland.com/anime-spotlight-visual-novels/

Steins;Gate [*]

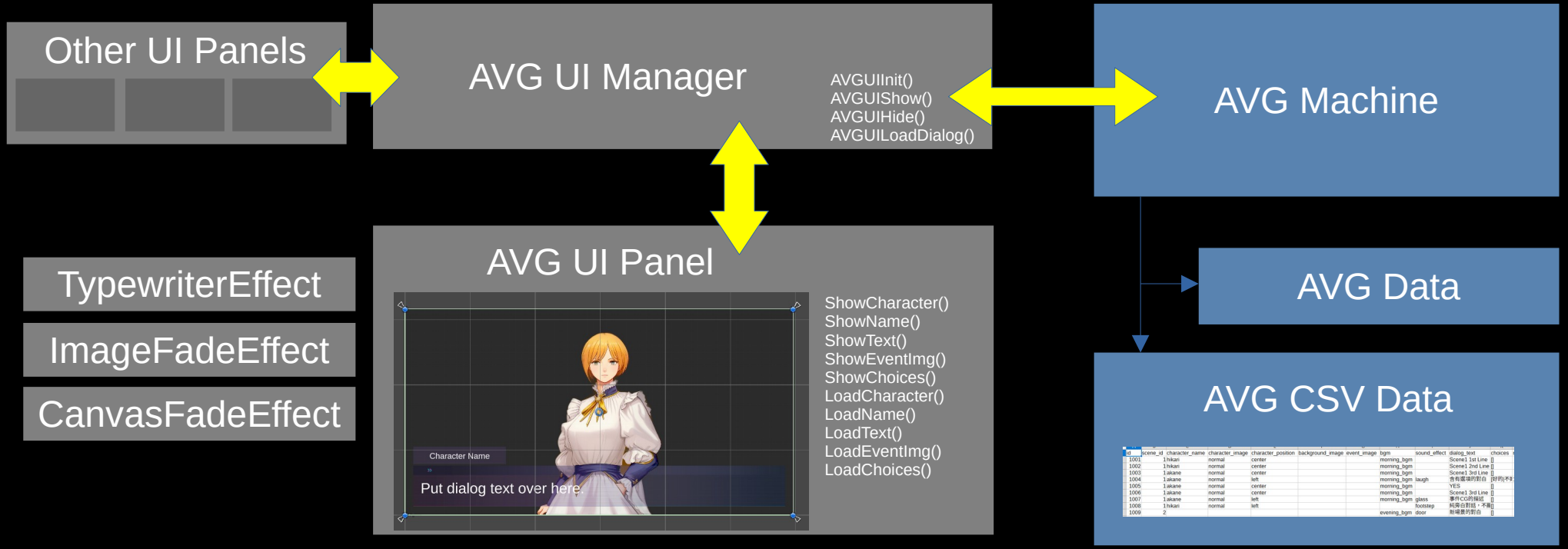
Phoenix Wright: Ace Attorney [*]

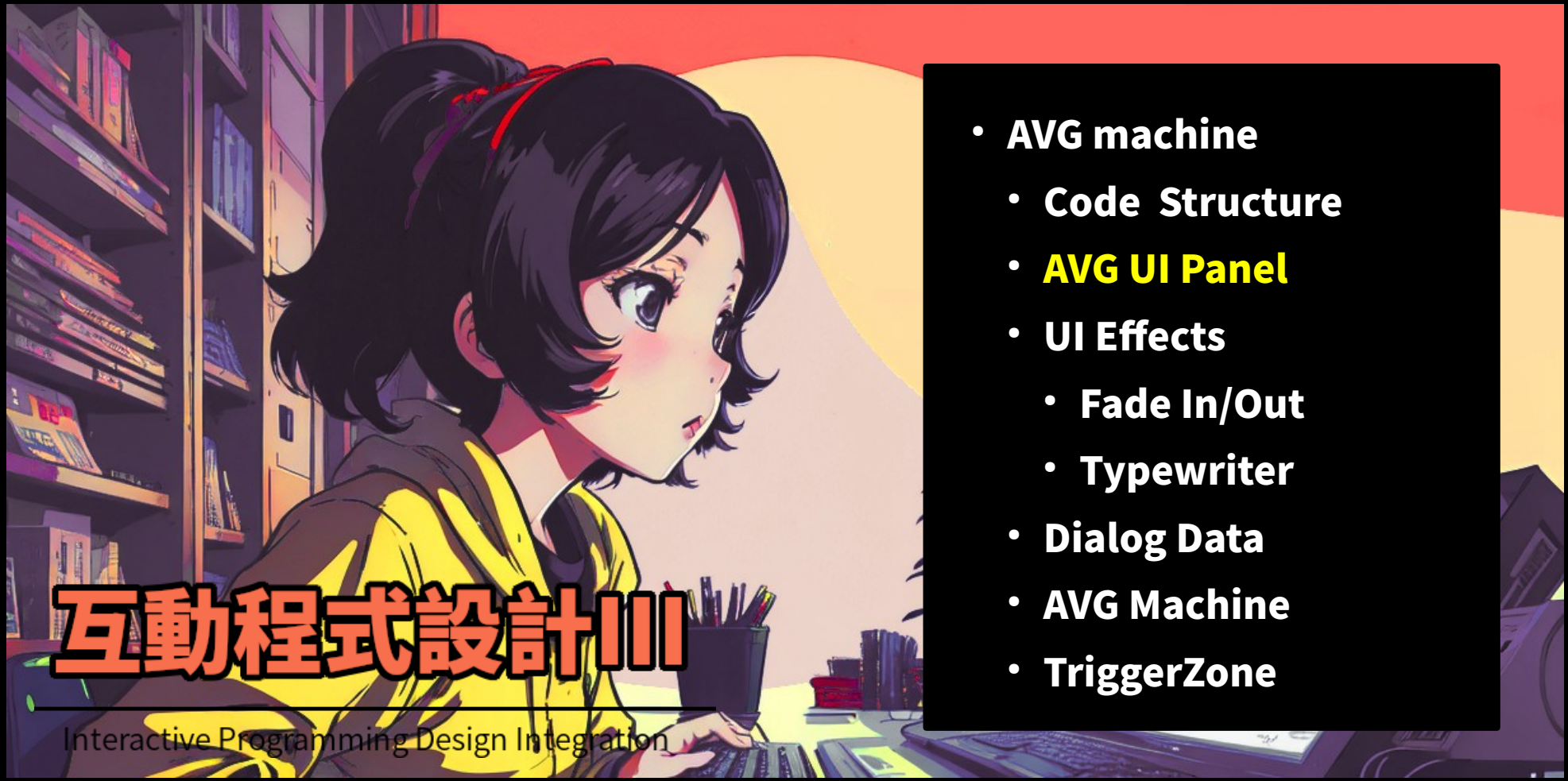


Danganronpa: Trigger Happy Havoc [*]

Code Structure

- We separate the design into two modules: UI & AVG
  - UI module: work as a standard UI. Focus at presenting a single dialog line.
  - AVG module: work as a flow control FSM. Focus at playing multiple dialog lines in predefined series and logics.

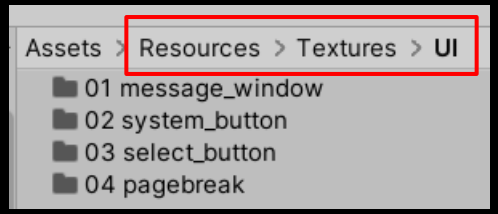


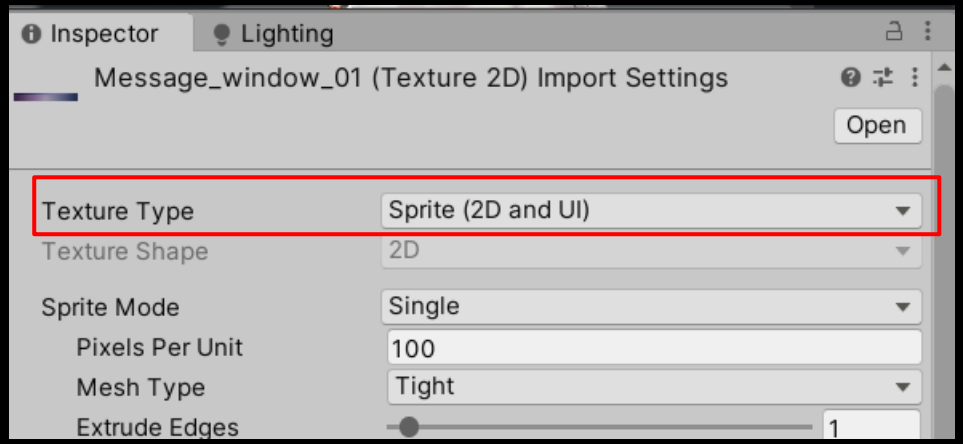


- AVG machine
  - Code Structure
  - AVG UI Panel
  - UI Effects
    - Fade In/Out
    - Typewriter
  - Dialog Data
  - AVG Machine
  - TriggerZone

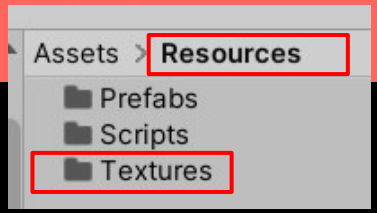
Free Assets

- 空想曲線：フリー素材：メッセージウィンドウ素材 その 52
  - https://kopacurve.blog.fc2.com/blog-entry-692.html
- Download the assets and put them into Resources/Textures/UI
- Change the texture type of all these images to Sprite.





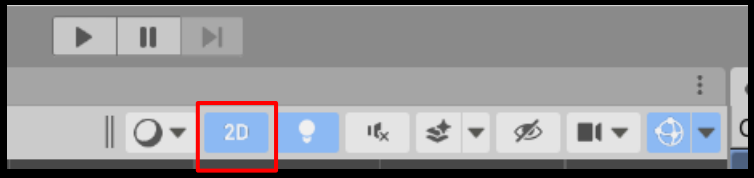
Import Assets

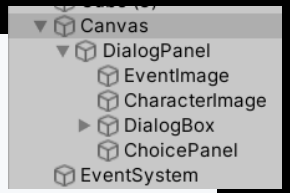


- Download the Resources08.zip and extract the Texture folder into Resources folder.
- Import settings:
  - Hikari: normal.png
    - Texture type: Sprite
    - Pixels per unit: 200
  - Akane: normal.png
    - Texture type: Sprite
    - Pixels per unit: 200
  - Events01: events_01.png/eventts_02.png
    - Texture type: Sprite
    - Pixels per unit: 200

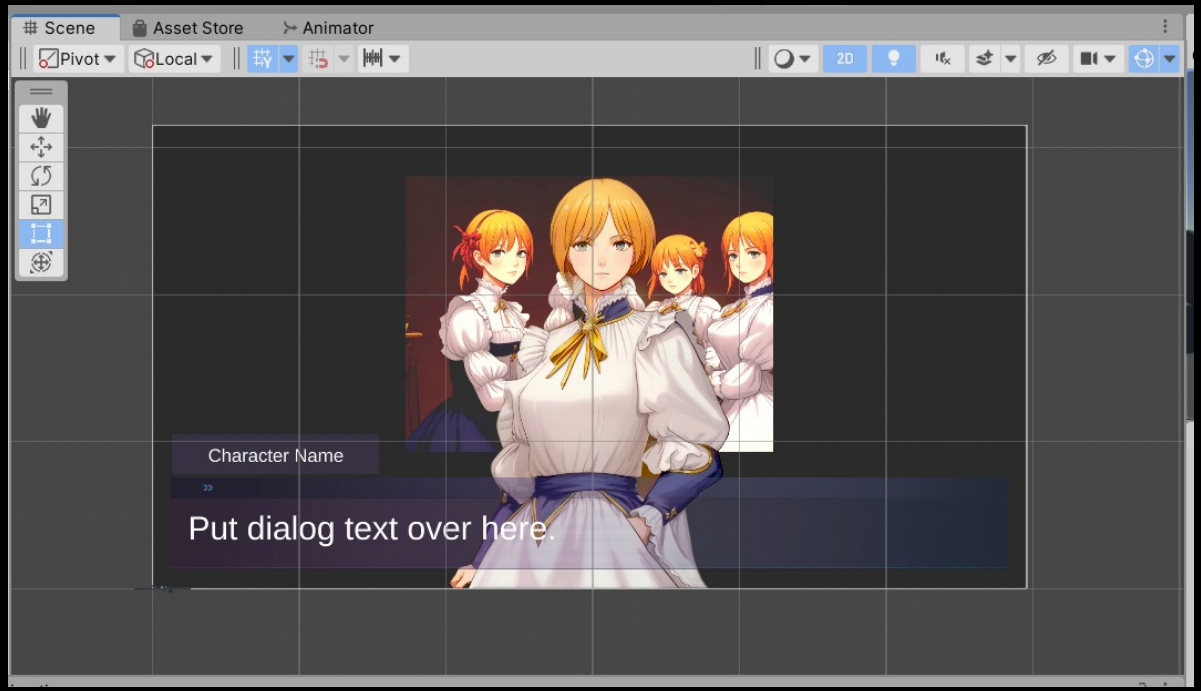
GameObject Structure

- Click the 2D button to switch camera view to 2D.
- Add a panel as DialogPanel and assemble the whole UI panel with your assets
  - Dialog Panel: bg color(0,0,0,100)
  - Character Image: Characters/hikari/normal.png
  - Event Image: Events/events_01.png
  - DialogBox: message_window_01.png
  - Name Panel: bg color(70,55,90,127)



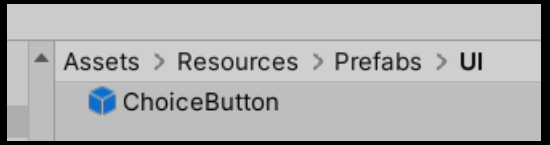



```
DialogPanel (Image)
├── CharacterImage (Image)
├── EventImage (Image)
├── DialogBox (Image)
│   ├── NamePanel (Image)
│   │   └── CharacterName (TextMeshPro)
│   └── DialogText (TextMeshPro)
└── ChoicePanel (VerticalLayoutGroup)
    └── Configuration:
        - Spacing: 10
        - Padding: 20
        - Child Alignment: Middle Center
```



Button Prefab

- Add a Button as a child of ChoicePanel. Name it ChoiceButton.
- Settings for ChoiceButton:
  - Width: 650
  - Height: 60
    - Image → source image: select_button
    - Button → Transition: Sprite Swap
    - Button → Highlighted Sprite: select_button_hover
    - Button → Pressed Sprite: select_button_click
    - Button → Selected Sprite: select_button
    - Button → Disabled Sprite: select_button
- Settings for Text(TMP)
  - Font size: 24
  - Vertex color: (255,255,255,255)
  - Align: center, middle
- Drag the ChoiceButton to Resources/Prefabs/UI to make it a prefab.
  - When you finished, delete the ChoiceButton in scene UI.



AVG UI Panel

- Add a AVGUIPanel class on DialogPanel GameObject:

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

namespace AVG
{
    public class AVGUIPanel : MonoBehaviour
    {
        [SerializeField] private GameObject dialogPanel;
        [SerializeField] private GameObject choicePanel;
        [SerializeField] private Image characterImage;
        [SerializeField] private Image eventImage;
        [SerializeField] private Image dialogBoxImage;
        [SerializeField] private Image namePanelImage;
        [SerializeField] private TextMeshProUGUI dialogBoxText;
        [SerializeField] private TextMeshProUGUI characterNameText;

        public void ShowCharacter(Sprite image,
            bool visibility = true, string position = "center")
        {
            characterImage.enabled = visibility;
            characterImage.sprite = image;
            characterImage.preserveAspect = true;
            switch (position) {
                case "left":
                    characterImage.rectTransform.anchoredPosition
                        = new Vector2(-300, 0);
                    break;
                case "right":
                    characterImage.rectTransform.anchoredPosition
                        = new Vector2(300, 0);
                    break;
                default:
                    characterImage.rectTransform.anchoredPosition
                        = new Vector2(0, 0);
                    break;
            }
        }

        public void ShowEvent(Sprite image, bool visibility = true) {
            eventImage.enabled = visibility;
            eventImage.sprite = image;
        }

        public void ShowDialogBox(string nameContent,
            string dialogContent, bool visibility = true)
        {
            dialogBoxImage.enabled = visibility;
            namePanelImage.enabled = visibility;
            dialogBoxText.enabled = visibility;
            characterNameText.text = nameContent;
            dialogBoxText.text = dialogContent;
        }

        public void ShowChoices(List<string> choices, List<int> nextSceneIDs,
            GameObject buttonPrefab, bool visibility)
        {
            choicePanel.SetActive(visibility);
            foreach (Transform button in choicePanel.transform) {
                Destroy(button.gameObject);
            }

            if (buttonPrefab == null) {
                return;
            }

            for (int i = 0; i < choices.Count; i++) {
                GameObject button = Instantiate(buttonPrefab);
                button.transform.parent = choicePanel.transform;
                button.transform.localScale = Vector3.one;
                button.GetComponentInChildren<TextMeshProUGUI>().text = choices[i];
            }
        }
    }
}
```

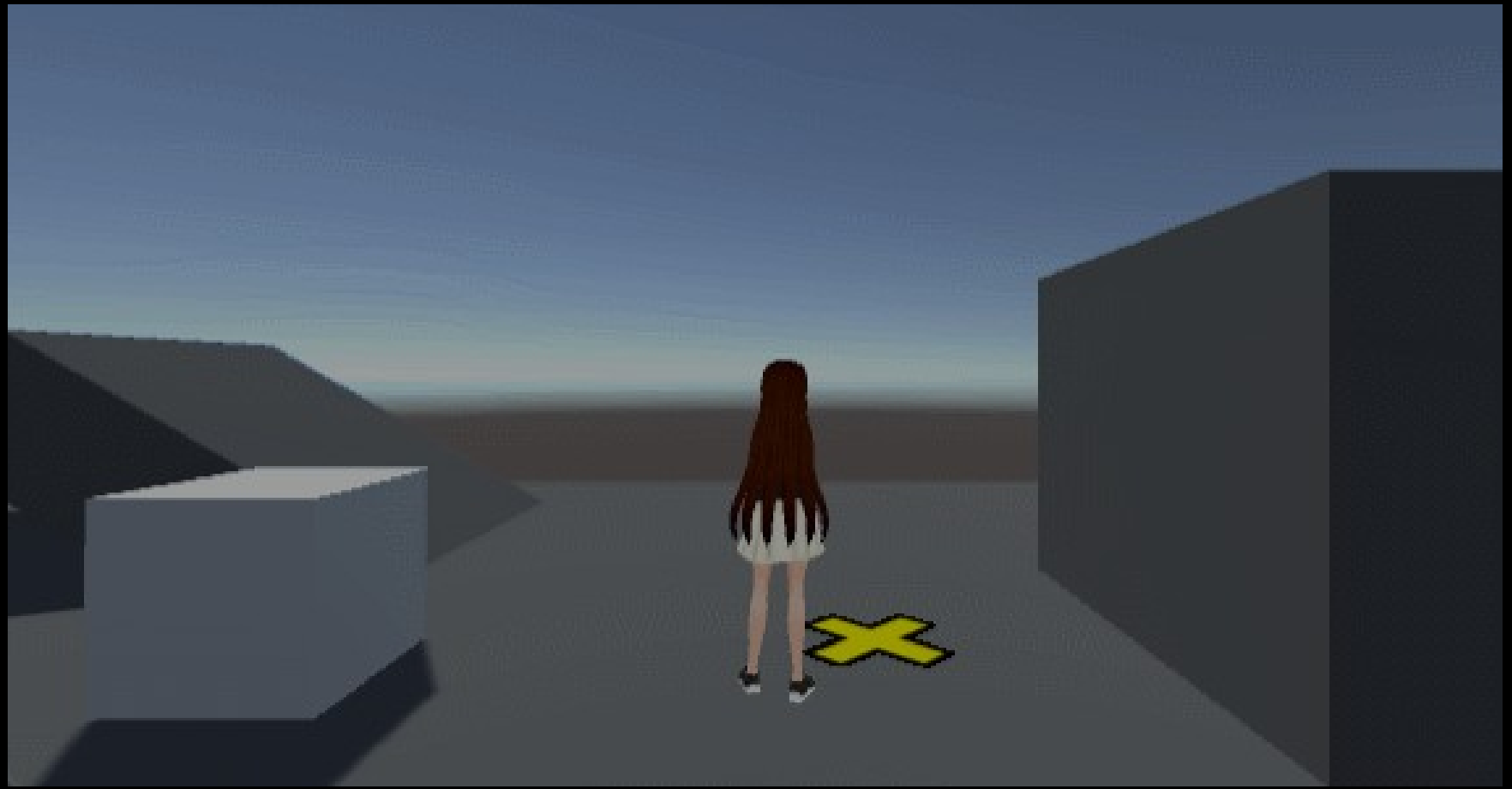
Test Client

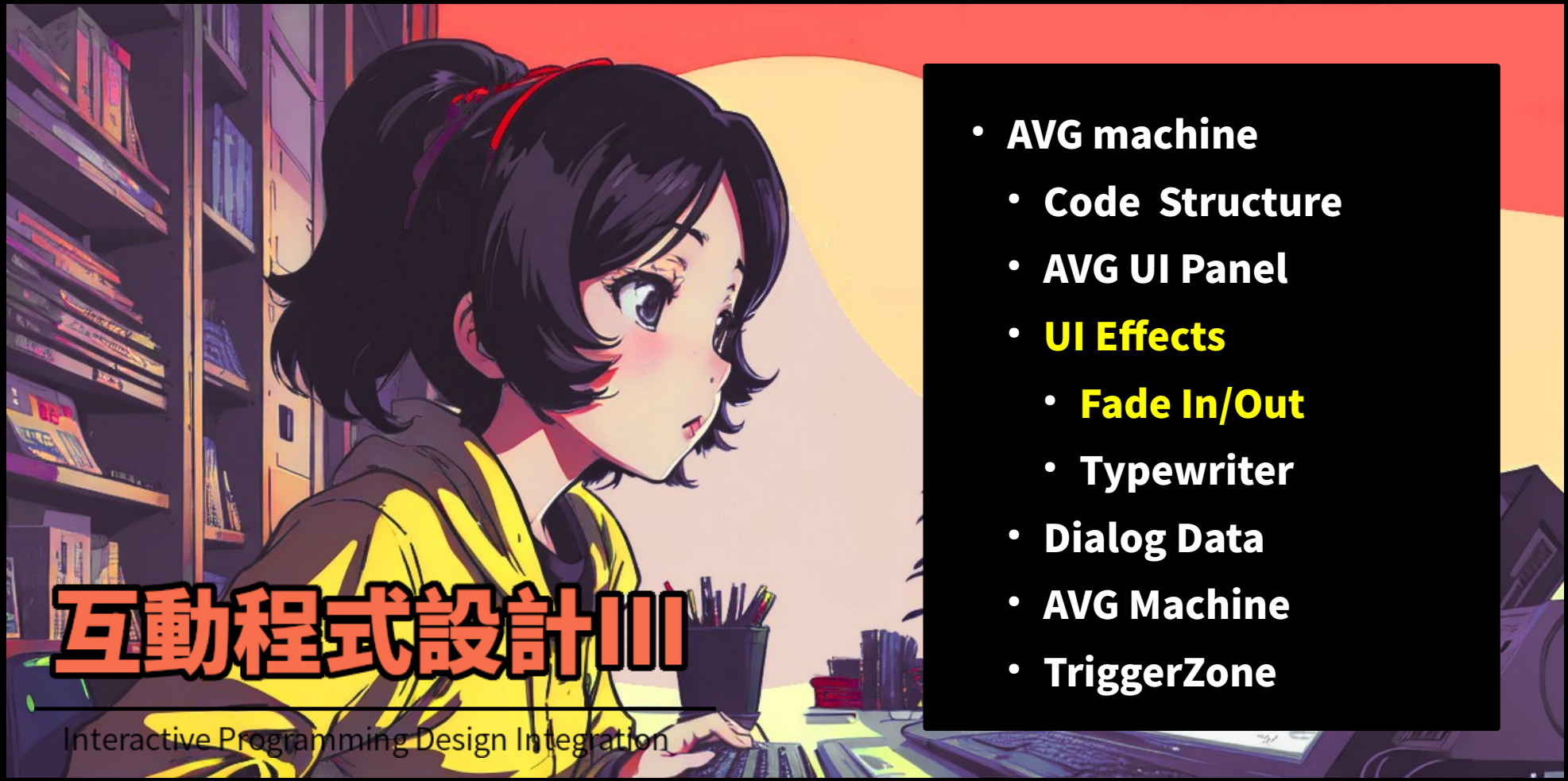
- Add a TestClient class on an empty GameObject. Name the GameObject TestClient:

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

public class TestClient : MonoBehaviour
{
    AVGUIPanel panel;
    Sprite hikari, akane, event01, event02;
    GameObject buttonPrefab;

    void Start()
    {
        panel = FindAnyObjectByType<AVGUIPanel>();
        hikari = Resources.Load<Sprite>("Textures/Characters/hikari/normal");
        akane = Resources.Load<Sprite>("Textures/Characters/akane/normal");
        event01 = Resources.Load<Sprite>("Textures/Events/event_01");
        event02 = Resources.Load<Sprite>("Textures/Events/event_02");
        buttonPrefab = Resources.Load<GameObject>("Prefabs/UI/ChoiceButton");
    }

    private void Update()
    {
        if (Input.GetKeyDown("1"))
        {
            panel.ShowCharacter(hikari);
        }
        if (Input.GetKeyDown("2"))
        {
            panel.ShowCharacter(akane);
        }
        if (Input.GetKeyDown("3"))
        {
            panel.ShowCharacter(null, false);
        }
        if (Input.GetKeyDown("4"))
        {
            panel.ShowDialogBox("akane", "Hello world.");
        }
        if (Input.GetKeyDown("5"))
        {
            panel.ShowDialogBox("hikari", "How are you?");
        }
        if (Input.GetKeyDown("6"))
        {
            panel.ShowDialogBox(null, null, false);
        }
        if (Input.GetKeyDown("7"))
        {
            panel.ShowEvent(event01);
        }
        if (Input.GetKeyDown("8"))
        {
            panel.ShowEvent(event02);
        }
        if (Input.GetKeyDown("9"))
        {
            panel.ShowEvent(null, false);
        }
        if (Input.GetKeyDown("0"))
        {
            panel.ShowChoices(new List<string> { "Yes", "No" }, new List<int> { 0, 0 }, buttonPrefab, true);
        }
        if (Input.GetKeyDown("-"))
        {
            panel.ShowChoices(null, null, null, false);
        }
    }
}
```

Test

- Test it. Press 1, 2, 3...0, "-" to check the UI panel functions.





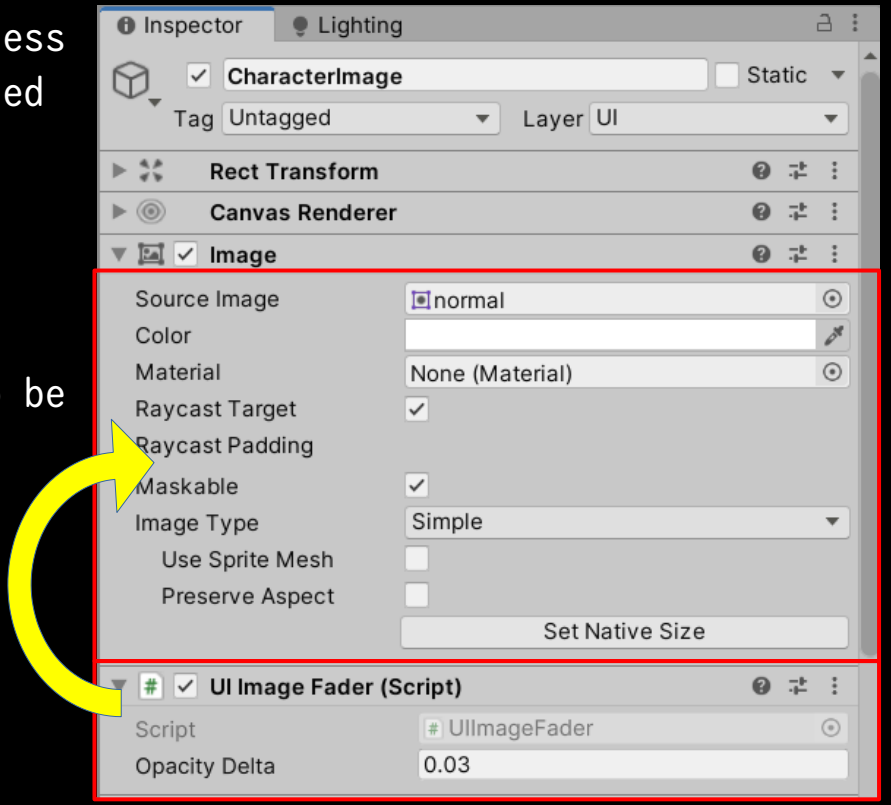
- AVG machine
  - Code Structure
  - AVG UI Panel
  - UI Effects
    - Fade In/Out
    - Typewriter
  - Dialog Data
  - AVG Machine
  - TriggerZone

Effect Moduels

- Effect moduels works as automated subsystems. It checks the current status of UI panel and edit it without noticing any other.
  - Autonomy in Operation
    - Effect modules act as self-contained automated subsystems.
    - They independently monitor and modify the current status of a UI panel without interacting with or depending on other modules.
  - Scope Isolation
    - Each effect focuses exclusively on its designated task or scope.
    - This ensures no conflicts occur when multiple effects operate on the same UI component.
  - Supporting Variants with Abstraction
    - To accommodate multiple effect variants, we use an abstract class to define and regulate interfacing methods, ensuring consistency and extensibility.

Fade In/Out

- Stateless Design
  - The Fade module will be implemented as a stateless component, ensuring no internal state is retained for better modularity and reusability.
- Key Property: opacityDelta
  - The module includes a property called opacityDelta.
  - This property defines the incremental change to be applied to the opacity of a sibling Image component, controlling the fade effect dynamically.



UI Image Fader

- Add a new script to the CharacterImage object. Name the script: UIImageFader.
- Fade behavior
  - The default value for the increment property is 0.04, enabling a fade-in effect by gradually increasing the image's opacity over time. ❶
  - To select between fade-in/out effect, set the speed to a positive of a negative value respectively.
  - Load the color value from image every tick and update the opacity channel. ❷ Clamp the opacity to [0..1] span.
  - TriggerFade sets the fade speed according dir parameter. ❸
  - StopFade sets the fade speed to 0. ❹

```
1  using UnityEngine;
2  using UnityEngine.UI;
3
4  public class UIImageFader : MonoBehaviour
5  {
6      private Image image;
7  ❶   public float opacityDelta = 0.04f;
8      private float speed = 0;
9
10     private void Awake()
11     {
12         image = GetComponent<Image>();
13     }
14
15     private void Update()
16     {
17         Color tempColor = image.color;
18 ❷       tempColor.a = Mathf.Clamp((tempColor.a + speed * opacityDelta), 0, 1.0f);
19         image.color = tempColor;
20     }
21
22     /// <summary>
23     /// Trigger CharacterImage to fade in or out.
24     /// </summary>
25     /// <param name="dir">ture: Fade in, false: Fade out.</param>
26     public void TriggerFade(bool dir, bool restart = false)
27     {
28         if (restart)
29         {
30             Color tempColor = image.color;
31             tempColor.a = (dir ? 0f : 1.0f);
32             image.color = tempColor;
33         }
34         speed = (dir ? 1.0f : -1.0f);
35     }
36
37     public void StopFade()
38     {
39         speed = 0;
40     }
41 }
```

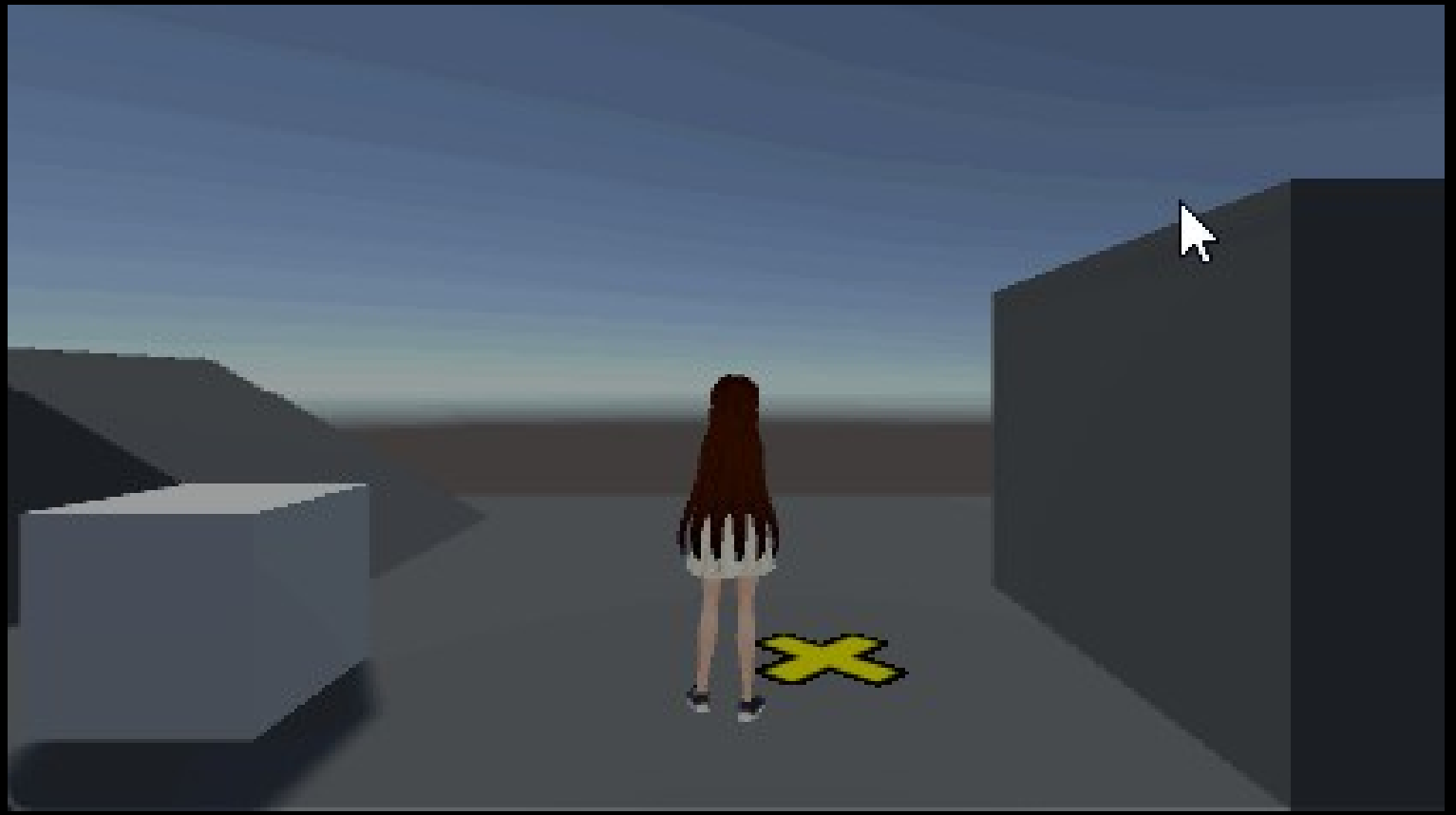
Edit AVGUIPanel

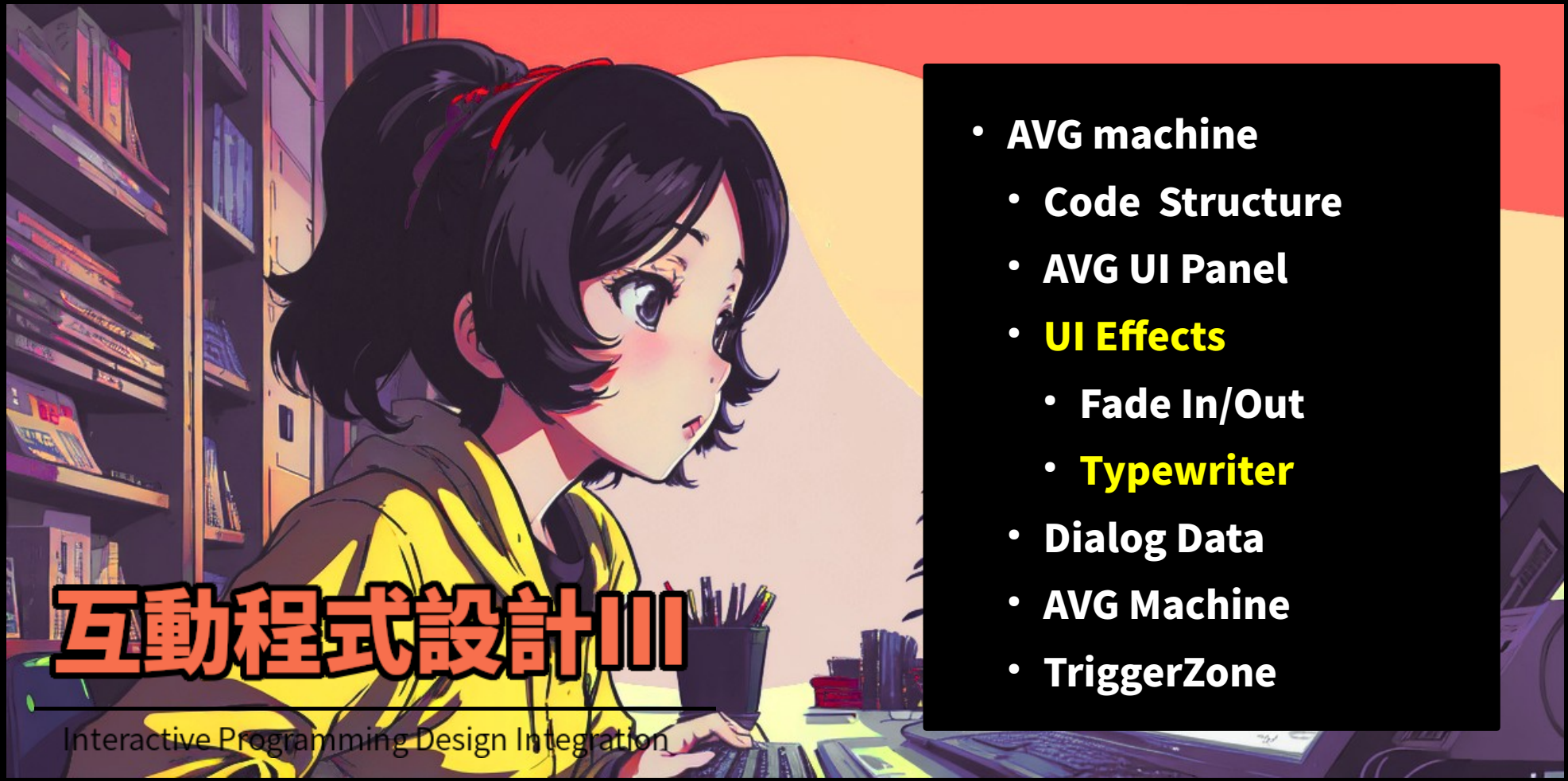
- Go back and modify AVGUIPanel. Get fader and call TriggerFade for both CharacterImage and EventImage. ❶❷

```
17      public void ShowCharacter(Sprite image,
18          bool visibility = true, string position = "center")
19      {
20          characterImage.enabled = visibility;
21          characterImage.sprite = image;
22          characterImage.preserveAspect = true;
23          switch (position) {
24              case "left":
25                  characterImage.rectTransform.anchoredPosition
26                      = new Vector2(-300, 0);
27                  break;
28              case "right":
29                  characterImage.rectTransform.anchoredPosition
30                      = new Vector2(300, 0);
31                  break;
32              default:
33                  characterImage.rectTransform.anchoredPosition
34                      = new Vector2(0, 0);
35                  break;
36          }
37
38          UIImageFader fader = characterImage.GetComponentInChildren<UIImageFader>();
39  ❶       fader.TriggerFade(true, true);
40      }
41
42      public void ShowEvent(Sprite image, bool visibility = true) {
43          eventImage.enabled = visibility;
44          eventImage.sprite = image;
45
46  ❷       UIImageFader fader = eventImage.GetComponentInChildren<UIImageFader>();
47          fader.TriggerFade(true, true);
48      }
```

Test

- Test it. Press 1, 2, 3...0, "-" to check the UI panel functions.

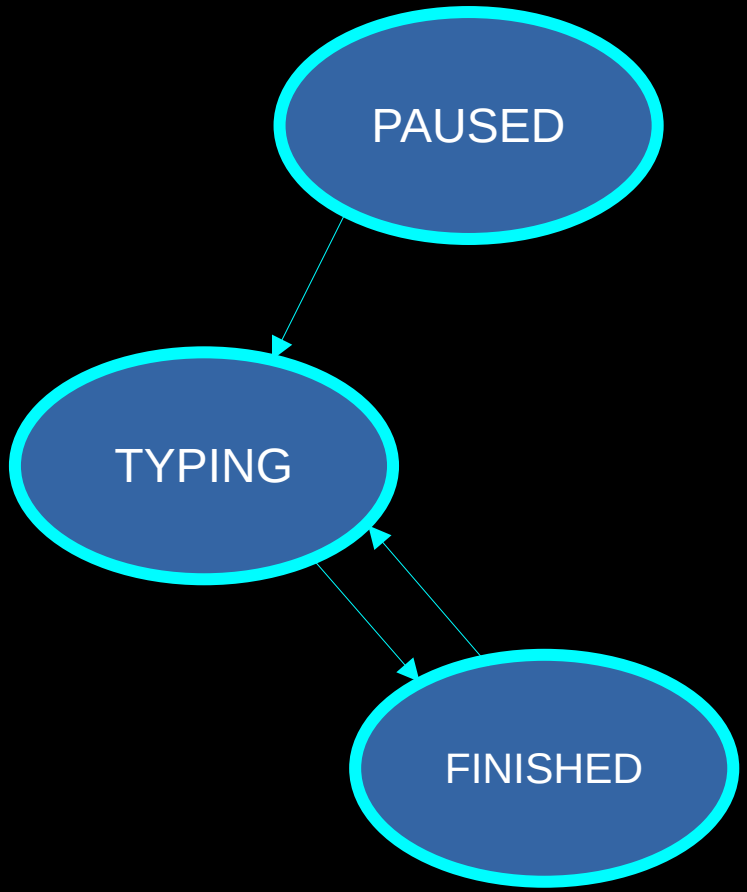




- AVG machine
  - Code Structure
  - AVG UI Panel
  - UI Effects
    - Fade In/Out
    - Typewriter
  - Dialog Data
  - AVG Machine
  - TriggerZone

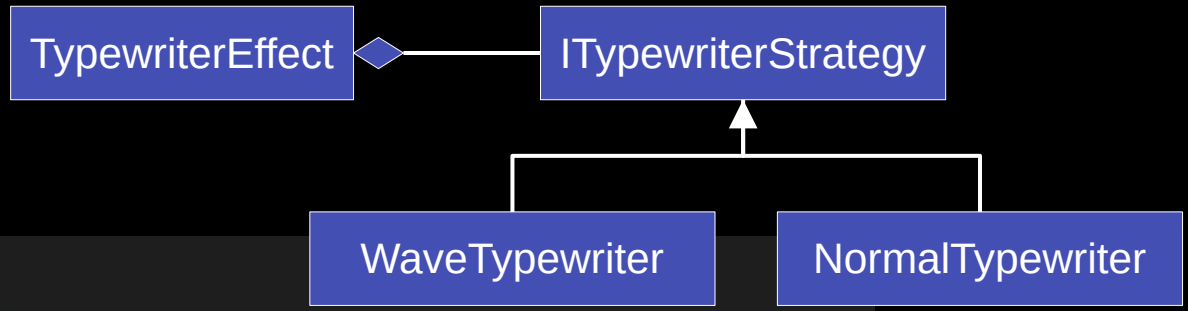
Typewriter

- Stateful Design with Finite State Machine (FSM)
  - Typewriter as an FSM
    - The typewriter operates as a Finite State Machine (FSM), enhancing stability and reducing unexpected bugs.
  - Typewriter Variants
    - Two types of typewriters inherit from a shared abstract interface:
      - Normal Typewriter: Displays the next character at regular intervals.
      - Wave Typewriter: Adds a dynamic effect where each new character drops in with a scaling-down font size.
  - Design Pattern
    - The Strategy Pattern is employed to manage the behavior variations, ensuring modular and maintainable code.



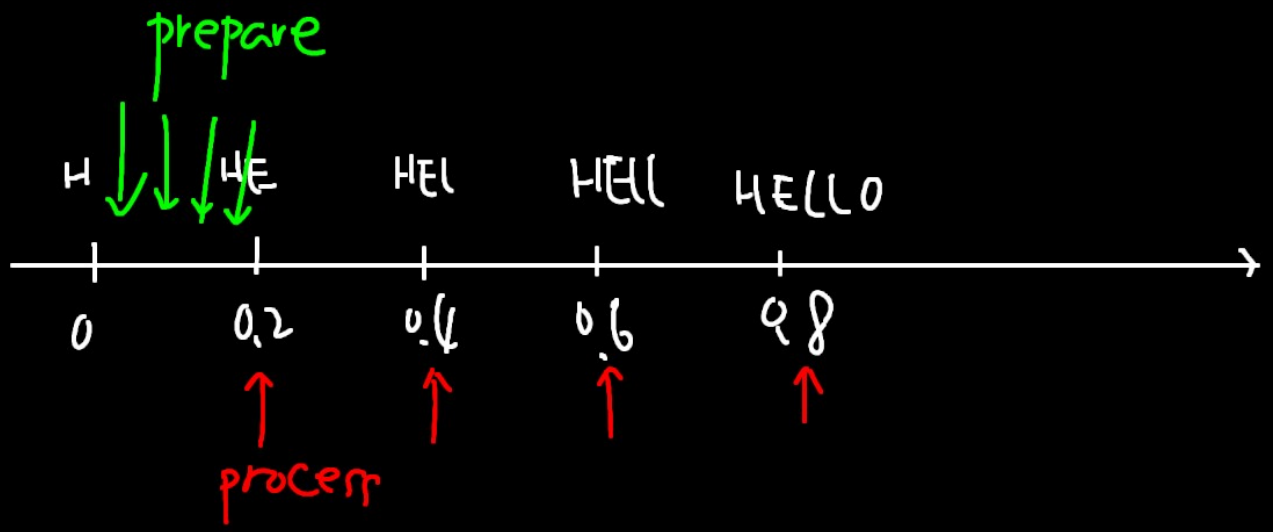
Abstract Strategy

- Key Methods
  - PrepareText(): Continuously called while waiting for the next character to be typed.
  - ProcessText(): Executes when it’s time to display the next character.
- Additional Functionalities
  - GetTypeSpeed(): Returns the time interval between typing two consecutive characters.
  - IsComplete(): Indicates whether the typing process is finished.




```
1   namespace AVG
2   {
3       public abstract class ITypewriterStrategy
4       {
5           public abstract string PrepareText(string fullText, int currentIndex, float elapsedTime);
6           public abstract string ProcessText(string fullText, int currentIndex, float elapsedTime);
7           public abstract bool IsComplete(int currentIndex, string fullText);
8           public abstract float GetTypeSpeed();
9       }
10  }
```

Prepare/Process

- 

Edit InputManager

- Enhancing Dialog Interaction
  - To enable user interaction with dialogs, a new event and corresponding task have been added to the InputManager class.
- Changes Made
  - New Event: evtDialogClick (of type UnityEvent<bool>). ❶
  - New Task: Abstract method CalculateDialogClick() to handle dialog click processing. ❷
- Integration
  - The CalculateDialogClick() method is called within the Update() method, ensuring it is part of the regular input calculation cycle. ❸

```
1  using UnityEngine.Events;
2  using UnityEngine;
3
4  public abstract class InputManager : MonoBehaviour
5  {
6      public UnityEvent<Vector3> evtDpadAxis;
7      public UnityEvent<bool> evtJump;
8      public UnityEvent<bool> evtRun;
9  ❶   public UnityEvent<bool> evtDialogClick;
10
11     protected abstract void CalculateDpadAxis();
12     protected abstract void CalculateJump();
13     protected abstract void CalculateRun();
14 ❷   protected abstract void CalculateDialogClick();
15     protected abstract void PostProcessDpadAxis();
16
17     private void Update()
18     {
19         CalculateDpadAxis();
20         CalculateJump();
21         CalculateRun();
22 ❸       CalculateDialogClick();
23         PostProcessDpadAxis();
24     }
25 }
```

Edit KeyboardInputManager

- Implementing the concrete CalculateDialogClick() method
  - Detects whether the left mouse button (mouse 0) is being pressed for once.
  - Passes the button press signal to the evtDialogClick event for further processing. ❶

```
1   using UnityEngine;
2   public class KeyboardInputManager : InputManager
3   {
4       private Vector3 axis;
5       private bool jump;
6       private bool run;
7       private bool dialogClick;
8
9       protected override void CalculateDpadAxis()...
31
32      protected override void CalculateJump()...
37
38      protected override void CalculateRun()...
43
44      protected override void CalculateDialogClick()
45      {
46  ❶       dialogClick = Input.GetKeyDown("mouse 0");
47          evtDialogClick?.Invoke(dialogClick);
48      }
49
50      protected override void PostProcessDpadAxis()...
54  }
55
```

Normal Typing Effect

- TypeSpeed Initialization
  - The typeSpeed can only be set in the constructor for immutability. ❶
- ProcessText and PrepareText
  - Both methods function identically: they display the portion of the string up to the current index. ❷
- GetTypeSpeed as a Getter
  - The GetTypeSpeed method is implemented as a getter. ❸
  - This simplifies the code by reducing redundancy, achieving the same functionality in a more concise manner.

```
1  namespace AVG
2  {
3      public class NormalTypewriter : ITypewriterStrategy {
4          private readonly float typeSpeed;
5
6          ❶ public NormalTypewriter(float typeSpeed = 0.05f) {
7              this.typeSpeed = typeSpeed;
8          }
9
10         ❷ public override string ProcessText(
11             string fullText, int currentIndex, float elapsedTime) {
12             return fullText.Substring(0, currentIndex);
13         }
14
15         public override string PrepareText(
16             string fullText, int currentIndex, float elapsedTime) {
17             return fullText.Substring(0, currentIndex);
18         }
19
20         public override bool IsComplete(
21             int currentIndex, string fullText) {
22             return currentIndex >= fullText.Length;
23         }
24
25         // This method is defined as a getter.
26         // It is equivalent to:
27         // public override float GetTypeSpeed() {
28         //     return typeSpeed;
29         // }
30         ❸ public override float GetTypeSpeed() => typeSpeed;
31     }
32 }
```

TypewriterEffect

```
using System;
using TMPro;
using UnityEngine;

namespace AVG {
    public class TypewriterEffect : MonoBehaviour {
        private enum STATE {
            PAUSED,
            TYPING,
            FINISHED
        }
        [SerializeField]
        private STATE state;

        private ITypewriterStrategy currentStrategy;
        private TextMeshProUGUI textComponent;
        private string fullText;
        private float elapsedTime;
        private int currentCharIndex;
        private Action onTypeComplete;
        private KeyboardInputManager inputManager;
        private bool triggerEnter;

        private void Awake() {
            textComponent = GetComponent<TextMeshProUGUI>();
            inputManager = FindObjectOfType<KeyboardInputManager>();
            inputManager.evtDialogClick.AddListener(OnDialogClicked);
            SetStrategy(new NormalTypewriter());
            GoToState(STATE.PAUSED);
        }
```

```
        private void OnDialogClicked(bool pressed) {
            if (this.enabled && pressed && state == STATE.TYPING) {
                CompleteTyping();
            }
        }

        public void SetStrategy(ITypewriterStrategy strategy) {
            currentStrategy = strategy;
        }

        public void StartTyping(string text, Action onComplete = null) {
            fullText = text;
            textComponent.text = "";
            currentCharIndex = 0;
            elapsedTime = 0;
            onTypeComplete = onComplete;
            GoToState(STATE.TYPING);
        }
```

```
50
51      private void Update() {
52          switch (state) {
53              case STATE.PAUSED:
54                  break;
55              case STATE.TYPING:
56                  elapsedTime += Time.deltaTime;
57                  if (elapsedTime >= currentStrategy.GetTypeSpeed()) {
58                      elapsedTime = 0;
59                      currentCharIndex += 1;
60                      if (currentCharIndex > fullText.Length) {
61                          GoToState(STATE.FINISHED);
62                      }
63                      else {
64                          textComponent.text = currentStrategy.ProcessText(fullText, currentCharIndex, elapsedTime);
65                      }
66                  }
67                  else {
68                      textComponent.text = currentStrategy.PrepareText(fullText, currentCharIndex, elapsedTime);
69                  }
70                  break;
71              case STATE.FINISHED:
72                  if (triggerEnter) {
73                      //isTyping = false;
74                      textComponent.text = currentStrategy.ProcessText(fullText, currentCharIndex - 1, elapsedTime);
75
76                      onTypeComplete?.Invoke();
77                      triggerEnter = false;
78                  }
79                  break;
80              default:
81                  break;
82          }
83          return;
84      }
```

```
86          public void CompleteTyping() {
87              currentCharIndex = fullText.Length + 1;
88              GoToState(STATE.FINISHED);
89              onTypeComplete?.Invoke();
90          }
91
92          public bool IsTyping => (state == STATE.TYPING);
93          public bool IsFinished => (state == STATE.FINISHED);
94
95          private void GoToState(STATE targetState) {
96              state = targetState;
97              triggerEnter = true;
98          }
99      }
100 }
```

-

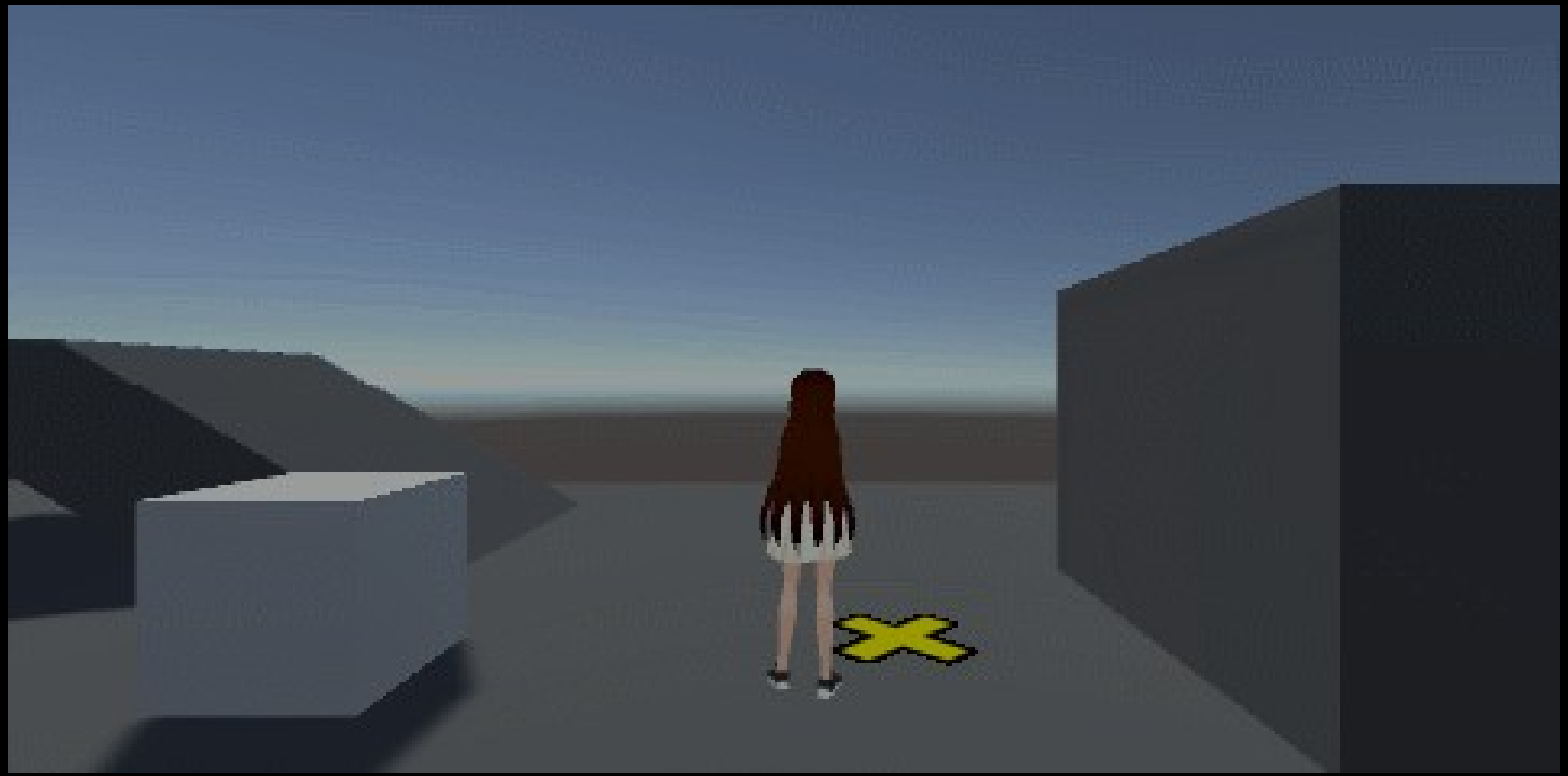
Edit AVGUIPanel

- Locate the ShowDialogBox Method
  - Find the ShowDialogBox method and modify it as follows.
- Integrates the Typewriter Effect:
  - Retrieves the TypewriterEffect component using GetComponentInChildren. ❶
  - If found, enables the typewriter and starts typing the dialogContent.

```
50      public void ShowDialogBox(string nameContent,
51          string dialogContent, bool visibility = true)
52      {
53          dialogBoxImage.enabled = visibility;
54          namePanelImage.enabled = visibility;
55          dialogBoxText.enabled = visibility;
56          characterNameText.text = nameContent;
57      ❶   //dialogBoxText.text = dialogContent;
58          TypewriterEffect typewriter = GetComponentInChildren<TypewriterEffect>();
59          if (typewriter != null) {
60              typewriter.enabled= visibility;
61              typewriter.StartTyping(dialogContent);
62          }
63      }
```

Test

- Test it. Press 4, 5, 6 to check the UI panel functions.



Wave Typing Effect

- Wave Typewriter Design
  - Implements the ITypewriterStrategy interface with a unique wave animation effect.
  - Uses a typeSpeed property (default: 0.05f) to control the typing interval.
  - Introduces a waveAmplitude property to adjust the intensity of the wave effect on characters. ❶
- PrepareText Method
  - Dynamically formats text to create the wave effect.
  - For each character being typed:
    - Applies a vertical offset (voffset) based on a sine wave function (Mathf.Sin).
    - Adjusts font size dynamically to simulate a scaling effect.

```
1   using UnityEngine;
2
3   namespace AVG
4   {
5       public class WaveTypewriter : ITypewriterStrategy
6       {
7           private readonly float typeSpeed;
8           private readonly float waveAmplitude;
9
10          public WaveTypewriter(
11  ❶           float typeSpeed = 0.05f, float amplitude = 0.5f)
12          {
13              this.typeSpeed = typeSpeed;
14              this.waveAmplitude = amplitude;
15          }
16
17          public override string ProcessText(
18              string fullText, int currentIndex, float elapsedTime)
19          {
20              // Play Audio effect here.
21              return fullText.Substring(0, currentIndex);
22          }
23
24          public override string PrepareText(
25              string fullText, int currentIndex, float elapsedTime)
26          {
27  ❷           if (currentIndex == 0)
28              {
29                  return $"<voffset={Mathf.Sin(elapsedTime * 10) * waveAmplitude}em>" +
30                      $"<size={elapsedTime * 2500}%>{fullText.Substring(0, 1)}</size>" +
31                      $"</voffset>";
32              }
```

Wave Typing Effect

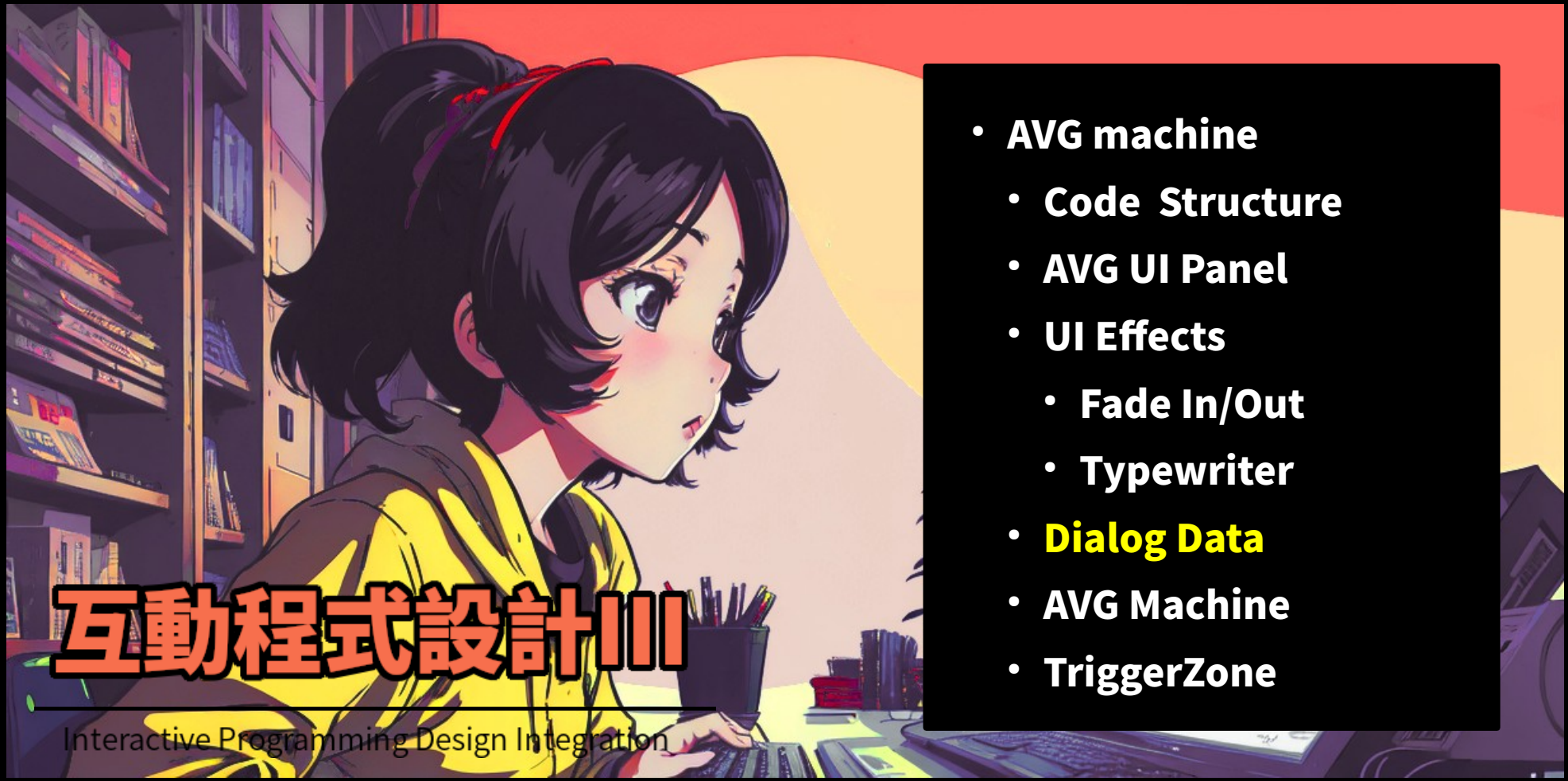
- Key Behaviors
  - Initial Character: (on last page)
  - If the currentIndex is 0, formats only the first character with wave animation. ❷
  - Intermediate Characters:
    - Displays typed text (textFront) while animating the current character (textBack) with a wave offset and size effect. ❸
  - Completion:
    - If the currentIndex exceeds the text length, displays the full text without effects. ❹
- IsComplete Methods
  - IsComplete(): Checks whether all characters have been typed (currentIndex >= fullText.Length). ❺
- Wave Effect Customization
  - Fine-tune the effect by modifying the waveAmplitude or typeSpeed to match the desired visual rhythm. ❻

```
33          ❸ else if (currentIndex < fullText.Length)
34            {
35                var textFront = fullText.Substring(0, currentIndex);
36                var textBack = fullText.Substring(currentIndex, 1);
37                //Add wave effect here.                                    ❻
38                //for typeSpeed = 0.1
39                //return $"{textFront}<voffset=" +
40                //    $"{Mathf.Sin(elapsedTime * 10 + Mathf.PI / 2) * waveAmplitude}" +
41                //    $"em><size={250 - elapsedTime * 2500}%>{textBack}</size></voffset>";
42
43                //for typeSpeed = 0.05
44                return $"{textFront}<voffset={Mathf.Sin(elapsedTime * 10) * waveAmplitude}em>" +
45                    $"<size={elapsedTime * 2500}%>{textBack}</size>" +
46                    $"</voffset>";
47            }
48          ❹ else
49            {
50                return fullText;
51            }
52        }
53
54        ❺ public override bool IsComplete(int currentIndex, string fullText)
55          {
56              return currentIndex >= fullText.Length;
57          }
58
59          public override float GetTypeSpeed() => typeSpeed;
60
61      }
62  }
```

Test

- Test it. Edit TypewriterEffect Awake to change from NormalTypewriter to WaveTypewriter. Press 4, 5, 6 to check the UI panel functions.

```
24      private void Awake() {
25          textComponent = GetComponent<TextMeshProUGUI>();
26          inputManager = FindObjectOfType<KeyboardInputManager>();
27          inputManager.evtDialogClick.AddListener(OnDialogClicked);
28          SetStrategy(new WaveTypewriter());
29          GoToState(STATE.PAUSED);
30      }
```



- AVG machine
  - Code Structure
  - AVG UI Panel
  - UI Effects
    - Fade In/Out
    - Typewriter
  - Dialog Data
  - AVG Machine
  - TriggerZone

Data Schema

- We are designing a data schema for dialog management. Each line of dialog in the conversation will be represented as an object of the DialogData class.
- Fields Explained
  - id: Unique identifier for the dialog line.
  - characterName: The name of the speaking character.
  - characterImage: Path or reference to the character's image.
  - characterPosition: Position of the character on the screen (e.g., left, center, right).
  - eventImage: Image related to an event, if applicable.
  - soundEffect: Sound effect to play during this line.
  - dialogText: The actual text of the dialog.
  - choices: a string of choices available to the player at this point. Use "|" as separator.
  - nextSceneIds: a string of IDs of the scenes to transition to based on choices. Use "|" as separator.
  - animation: Animation effect applied to the dialog or character.
  - voiceClip: Voice-over clip associated with the dialog.
  - displayType: Specifies how the dialog is displayed (e.g., narration, choice, event).

	A	B	C	D	E	F	G	H	I	J	K	L
1	id	character_name	character_expres	character_position	event_image	sound_effect	dialog_text	choices	next_scene_id	animation	voice_clip	display_type
2	1001	hikari	normal	center			Scene1 1st Line	[]			voice_1001	normal
3	1002	hikari	normal	center			Scene1 2nd Line	[]			voice_1001	normal
4	1003	akane	normal	center			Scene1 3rd Line	[]			voice_1001	normal
5	1004	akane	normal	left		laugh	Choices	[OK|No good]	1005|1008	shake	voice_1002	choice
6	1005	akane	normal	center			YES	[]			voice_1001	normal
7	1006	akane	normal	center			Scene1 3rd Line	[]			voice_1001	normal
8	1007	akane	normal	left	event_01	glass	Event	[]	1005	fade		event
9	1008	hikari	normal	left		footstep	Monologue	[]	1006			narration
10	1009	hikari	normal	center		door	End	[]	0		voice_1005	normal

DialogData

- Create a DialogData Class
  - Define a new class named DialogData and include all necessary fields for dialog management.
- Parsing Complex Fields
  - For fields like choices and nextSceneIds, parse the string data into appropriate list structures (List<string> and List<int>).
- Serialization for Unity
  - Add the [System.Serializable] attribute to make the DialogData class editable and viewable in the Unity Editor.

```
1   using System.Collections.Generic;
2
3   namespace AVG {
4       [System.Serializable]
5       public class DialogData
6       {
7           public int id;
8           public string characterName;
9           public string characterExpression;
10          public string characterPosition;
11          public string eventImage;
12          public string soundEffect;
13          public string dialogText;
14          public List<string> choices;
15          public List<int> nextSceneIds;
16          public string animation;
17          public string voiceClip;
18          public string displayType;
19      }
20  }
```

Singleton

```
using UnityEngine;

public abstract class Singleton<T> : MonoBehaviour where T : Singleton<T> {
    private static T instance;

    [SerializeField] private bool dontDestroyOnLoad = true;

    public static T Instance {
        get {
            if (instance == null) {
                instance = FindObjectOfType<T>();

                if (instance == null) {
                    GameObject go = new GameObject($"{typeof(T).Name}");
                    instance = go.AddComponent<T>();
                }
            }
            return instance;
        }
    }

    protected virtual void Awake() {
        if (instance != null && instance != this) {
            Destroy(gameObject);
            return;
        }

        instance = (T)this;

        if (dontDestroyOnLoad) {
            DontDestroyOnLoad(gameObject);
        }
        Init();
    }

    protected virtual void Init() { }
}
```

- What is a Singleton?
  - A Singleton ensures that a class has only one instance and provides a global point of access to it.
  - Commonly used in Unity for managing shared systems like game managers, input handlers, or audio controllers.
- Generic Implementation
  - The Singleton<T> class is a generic implementation, making it reusable for different types.
- Lazy Initialization
  - The Instance property checks if an instance exists; if not, it creates one using FindObjectOfType or adds the component to a new GameObject.
- Prevent Duplicate Instances
  - In the Awake method, if a duplicate instance is detected, it destroys the extra GameObject to maintain a single instance.
- Persist Across Scenes
  - The dontDestroyOnLoad flag determines whether the singleton persists across scene loads. If true, DontDestroyOnLoad is applied.

AVGUIManager

- Setup
  - Create a new GameObject named AVGManager.
  - Add a new class called AVGUIPanelManager and attach it to the GameObject.
- Singleton Design
  - The AVGUIPanelManager is implemented as a singleton for easy global access and interaction.
- Initialization
  - The Init method locates and assigns the AVGUIPanel and loads the choicePrefab from resources.
- Core Methods
  - AVGUIShow()
    - Activates the panel to display dialog elements.
  - AVGUIHide()
    - Hides the panel by deactivating it.
  - AVGUILoadDialog(DialogData dialog)
    - Processes the DialogData object and updates the UI based on the displayType field.
  - IsDialogComplete()
    - Check if the dialog is finished typing.
- DisplayType Handling
  - The switch statement handles different dialog display types:
    - Normal: Displays the character's image and dialog text.
    - Event: Shows event-related images and dialog.
    - Choice: Displays dialog options using choicePrefab.
    - Narration: Displays narration without additional elements.

```
using UnityEngine;

namespace AVG
{
    public class AVGUIManager : Singleton<AVGUIManager>
    {
        private GameObject choicePrefab;

        AVGUIPanel panel;

        protected override void Init() {
            panel = FindAnyObjectByType<AVGUIPanel>();
            choicePrefab = Resources.Load<GameObject>(
                "Prefabs/UI/ChoiceButton");
        }

        public void AVGUIShow() {
            panel.gameObject.SetActive(true);
        }

        public void AVGUIHide() {
            panel.gameObject.SetActive(false);
        }

        public void AVGUILoadDialog(DialogData dialog) {
            Sprite characterImage, eventImage;
            string dialogContent;

            switch (dialog.displayType) {
                case "normal":
                    characterImage = Resources.Load<Sprite>(
                        $"Textures/Characters/{dialog.characterName}/{dialog.characterExpression}");
                    dialogContent = dialog.dialogText;
                    panel.ShowCharacter(characterImage, true, dialog.characterPosition);
                    panel.ShowEvent(null, false);
                    panel.ShowDialogBox(dialog.characterName, dialogContent);
                    panel.ShowChoices(null, null, null, false);
                    break;
                case "event":
                    eventImage = Resources.Load<Sprite>(
                        $"Textures/Events/{dialog.eventImage}");
                    dialogContent = dialog.dialogText;
                    panel.ShowCharacter(null, false);
                    panel.ShowEvent(eventImage, true);
                    panel.ShowDialogBox("", dialogContent);
                    panel.ShowChoices(null, null, null, false);
                    break;
                case "choice":
                    panel.ShowCharacter(null, false);
                    panel.ShowEvent(null, false);
                    panel.ShowDialogBox(null,null,false);
                    panel.ShowChoices(dialog.choices, dialog.nextSceneIDs, choicePrefab, true);
                    break;
                case "narration":
                    dialogContent = dialog.dialogText;
                    panel.ShowCharacter(null, false);
                    panel.ShowEvent(null, false);
                    panel.ShowDialogBox("", dialogContent);
                    panel.ShowChoices(null, null, null, false);
                    break;
                default:
                    break;
            }
        }

        public bool IsDialogComplete()
        {
            TypewriterEffect typewriter = panel.GetComponentInChildren<TypewriterEffect>();
            return typewriter.IsFinished;
        }
    }
}
```

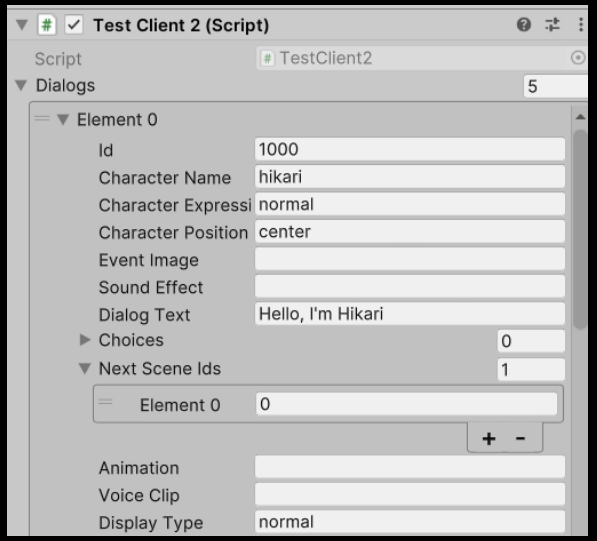
TestClient2

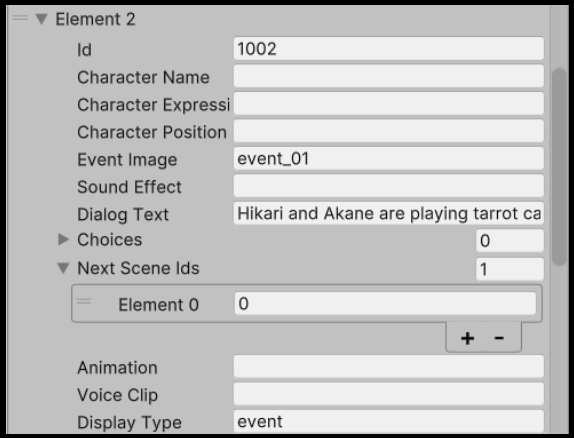
- Create a TestClient2 Class.
- Load the dependencies in Start.
- Load different dialogs into AVGUIpanel and check the responses.

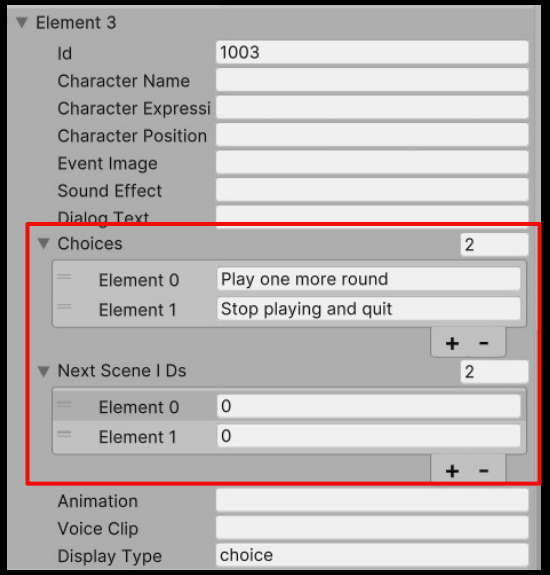
```
using AVG;
using System.Collections.Generic;
using UnityEngine;

public class TestClient2 : MonoBehaviour
{
    public List<DialogData> dialogs;

    AVGUIPanel panel;
    Sprite hikari, akane, event01, event02;
    GameObject buttonPrefab;

    void Start()
    {
        panel = FindAnyObjectByType<AVGUIPanel>();
        hikari = Resources.Load<Sprite>("Textures/Characters/hikari/normal");
        akane = Resources.Load<Sprite>("Textures/Characters/akane/normal");
        event01 = Resources.Load<Sprite>("Textures/Events/event_01");
        event02 = Resources.Load<Sprite>("Textures/Events/event_02");
        buttonPrefab = Resources.Load<GameObject>("Prefabs/UI/ChoiceButton");
        AVGUIManager.Instance.AVGUIHide();
    }

    void Update()
    {
        if (Input.GetKeyDown("1"))
        {
            AVGUIManager.Instance.AVGUIShow();
            DialogData dialog = dialogs[0];
            AVGUIManager.Instance.AVGUILoadDialog(dialog);
        }
        if (Input.GetKeyDown("2"))
        {
            AVGUIManager.Instance.AVGUIShow();
            DialogData dialog = dialogs[1];
            AVGUIManager.Instance.AVGUILoadDialog(dialog);
        }
        if (Input.GetKeyDown("3"))
        {
            AVGUIManager.Instance.AVGUIShow();
            DialogData dialog = dialogs[2];
            AVGUIManager.Instance.AVGUILoadDialog(dialog);
        }
        if (Input.GetKeyDown("4"))
        {
            AVGUIManager.Instance.AVGUIShow();
            DialogData dialog = dialogs[3];
            AVGUIManager.Instance.AVGUILoadDialog(dialog);
        }
        if (Input.GetKeyDown("5"))
        {
            AVGUIManager.Instance.AVGUIShow();
            DialogData dialog = dialogs[4];
            AVGUIManager.Instance.AVGUILoadDialog(dialog);
        }
        if (Input.GetKeyDown("6"))
        {
            AVGUIManager.Instance.AVGUIHide();
        }
    }
}
```

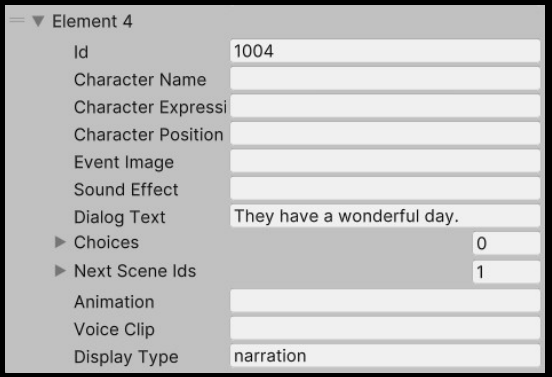
Test

- Attach TestClient2 to the TestClient GameObject.
- Add 5 Dialogs with the following details.



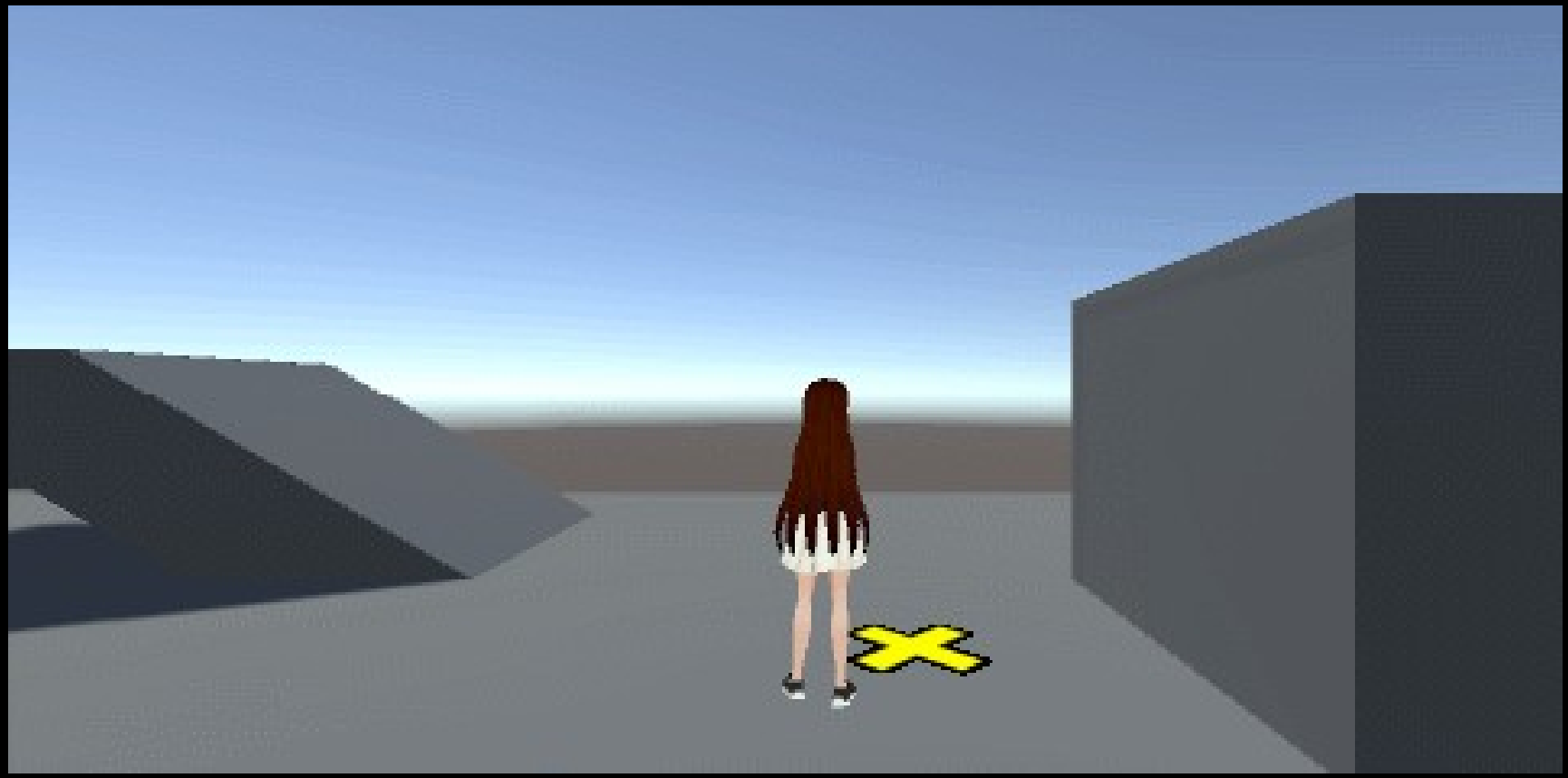


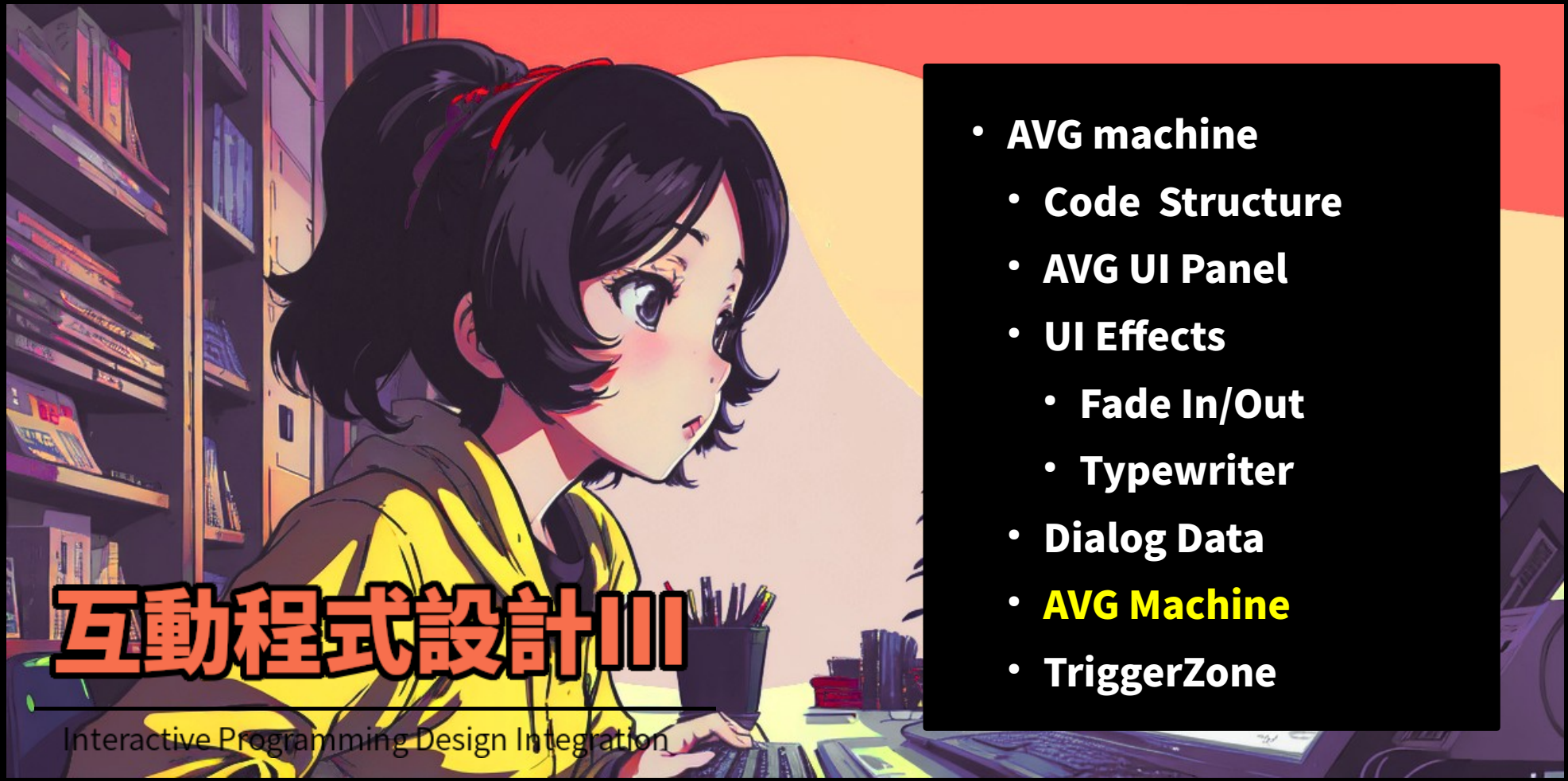




Test

- Test it. Press | ~ 6 to check the AVGUIManager functions.

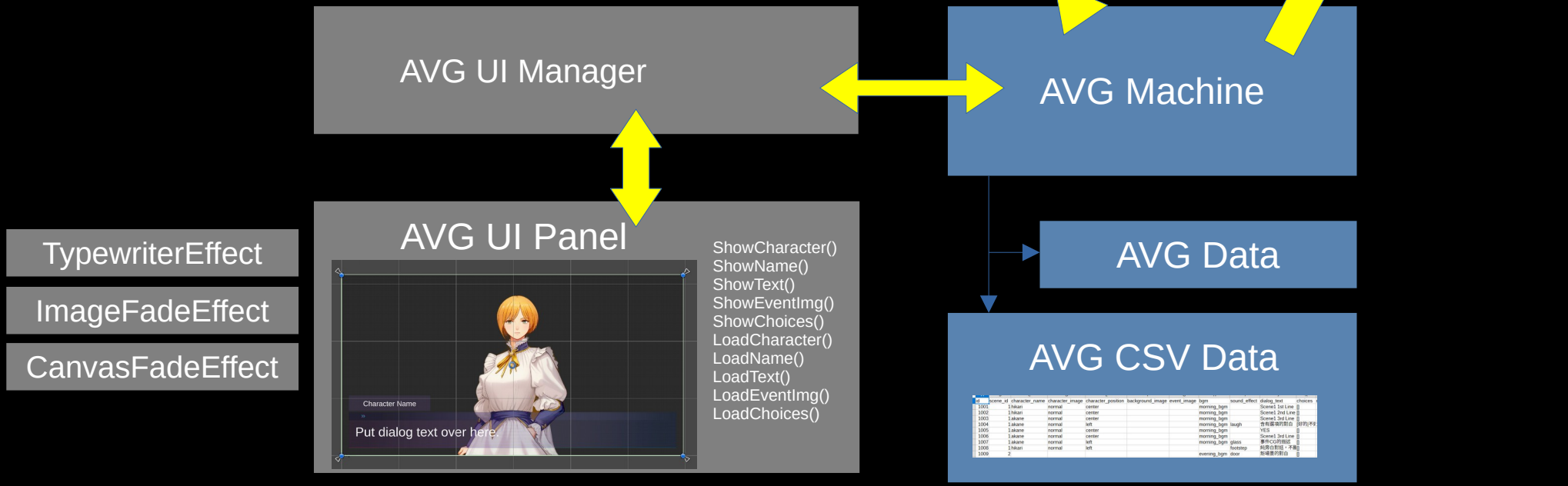




- AVG machine
  - Code Structure
  - AVG UI Panel
  - UI Effects
    - Fade In/Out
    - Typewriter
  - Dialog Data
  - AVG Machine
  - TriggerZone

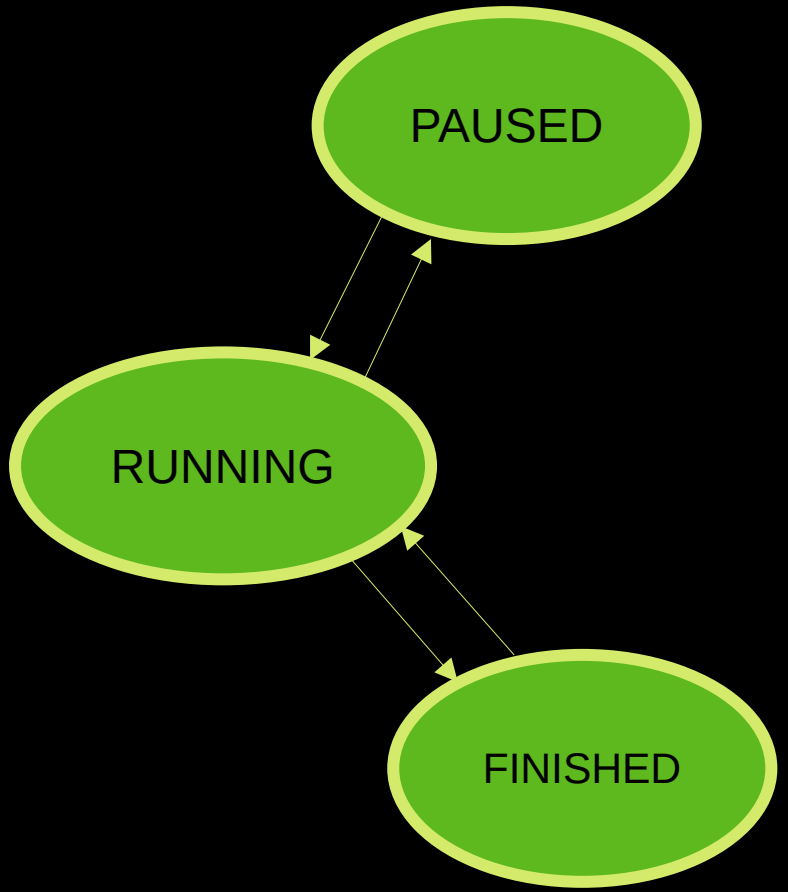
AVGMachine

- So far we can just load one line of dialog into the AVGUIPanel.
- With the help of AVGMachine, we can:
  - Load consecutive dialog lines.
  - Branching according to dialog line setting.
  - Interact with 3D trigger.
  - Extend the system to call method on other scripts.



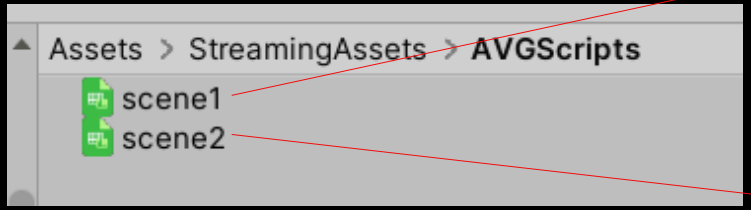
AVGMachine

- Core Functionality
  - The AVGMachine is responsible for managing and chaining dialog sequences, allowing both linear progression and non-linear jumps between dialogs.
- Singleton Design with FSM (Finite State Machine)
  - States:
    - PAUSED: DialogBox clicks are disabled. Only allows choice button clicks.
    - RUNNING: Enables user clicks to process dialog lines in both linear and non-linear ways.
    - FINISHED: Waits for the final user click to terminate the dialog session.
- Key Methods
  - LoadFromCSV: Loads dialog scripts from a CSV file.
  - Play: Starts playing the dialog sequence.
  - Stop: Stops the current dialog sequence.
  - GetCurrentDialog / GetNextDialog / GetDialogById: Retrieves a specific dialog entry.
  - SetNextDialog: Configures and initiates the next dialog in the sequence.



External CVS Dialog Scripts

- Locate the CSV Files
  - Extract the Resources08.zip file and find the scene1.csv and scene2.csv files.
  - !important! commas are NOT allowed to be put in any field in the contents!
  - If you want to enable commas in your dialog content, you can try to use excel sheet as a self-learning target. (or ask GPT to edit your codes.)
- Organize the Files
  - Place both CSV files in the StreamingAssets/AVGScripts folder.
- Accessing StreamingAssets
  - The StreamingAssets folder can be accessed programmatically in C# using:
    - Application.streamingAssetsPath

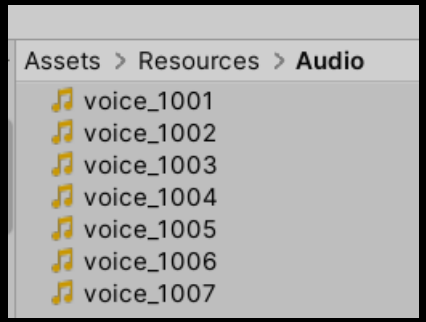


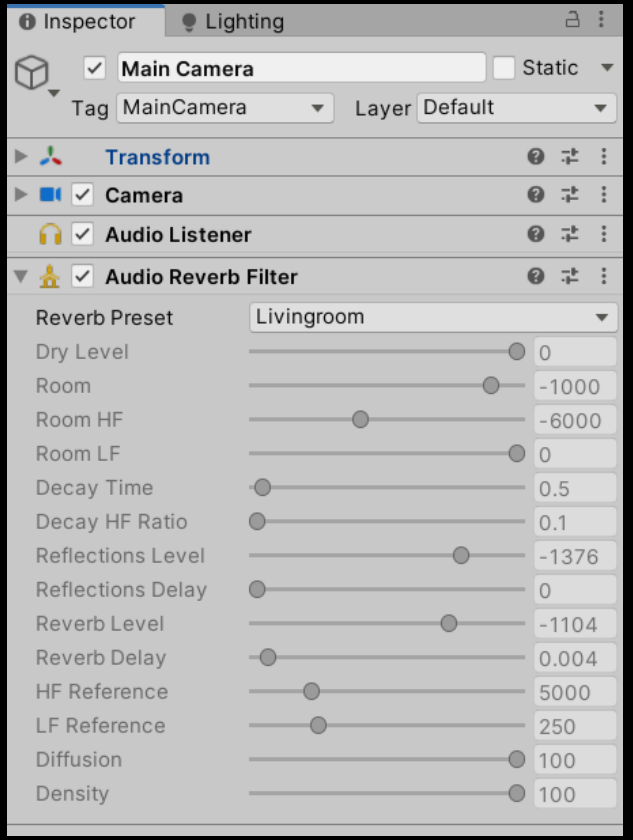
	A	B	C	D	E	F	G	H	I	J	K	L
1	id	character_name	character_expression	character_position	event_image	sound_effect	dialog_text	choices	next_scene_id	animation	voice_clip	display_type
2	1001	hikari	normal	center			Scene1 1st Line	[]			voice_1001	normal
3	1002	hikari	normal	center			Scene1 2nd Line	[]			voice_1001	normal
4	1003	akane	normal	center			Scene1 3rd Line	[]			voice_1001	normal
5	1004	akane	normal	left		laugh	Choices	[OK|No good]	1005|1009	shake	voice_1002	choice
6	1005	akane	normal	center			YES	[]			voice_1001	normal
7	1006	akane	normal	center			Scene1 3rd Line	[]			voice_1001	normal
8	1007	akane	normal	left	event_01	glass	Event	[]		fade		event
9	1008						Happy End		0			narration
10	1009	hikari	normal	left		footstep	Monologue	[]				narration
11	1010	hikari	normal	center		door	Bad End	[]	0		voice_1005	normal

	A	B	C	D	E	F	G	H	I	J	K	L
1	id	character_name	character_image	character_position	event_image	sound_effect	dialog_text	choices	next_scene_id	animation	voice_clip	display_type
2	1001	hikari	normal	center			Scene2 starts.				voice_1001	normal
3	1002	hikari	normal	right		laugh	Make choice…	[YES|NO]	1003|1005	shake	voice_1002	choice
4	1003	hikari	normal		event_02	glass	Drink coffee with friend.			fade		event
5	1004	hikari	normal			footstep	Happy End.		0			normal
6	1005	hikari	normal	left		door	Bad End		0		voice_1005	normal

Audio Files

- Locate the Audio Files
  - Extract the Resources08.zip file and find the mp3 files. Put them in Resources/Audio folder.
  - Add a Audio Reverb Filter on Main Camera. Set the preset to Livingroom





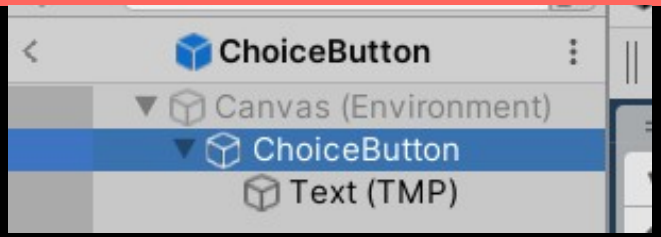
```csharp
using System.Collections.Generic;
using System.IO;
using System.Linq;
using UnityEngine;

namespace AVG
{
    public class AVGMachine : Singleton<AVGMachine>
    {
        private enum STATE {
            PAUSED,
            RUNNING,
            FINISHED
        }
        private STATE state;
        private bool triggerEnter;
        private int currentID;
        private InputManager inputManager;
        [SerializeField]
        List<DialogData> dialogs;

        protected override void Init() {
            currentID = 0;
            GoToState(STATE.FINISHED);
            inputManager = FindAnyObjectByType<InputManager>();
            inputManager.evtDialogClick.AddListener(OnDialogClicked);
        }

        private void OnDialogClicked(bool pressed)
        {
            // Accept this signal under RUNNING
            if (state == STATE.RUNNING) {
                // Reject this signal when typewriter is typing
                if (!AVGUIManager.Instance.IsDialogComplete()) {
                    return;
                }
                DialogData dialog = GetNextDialog();
                if (pressed) {
                    if (dialog != null) {
                        SetNextDialog(dialog);
                    }
                    else {
                        GoToState(STATE.FINISHED);
                    }
                }
            }
        }

        private void Update()
        {
            switch (state) {
                case STATE.PAUSED:
                    break;
                case STATE.RUNNING:
                    if (triggerEnter) {
                        AVGUIManager.Instance.AVGUIShow();
                    }
                    break;
                case STATE.FINISHED:
                    if (triggerEnter) {
                        AVGUIManager.Instance.AVGUIHide();
                    }
                    break;
                default:
                    break;
            }
        }

        private void GoToState(STATE targetState)
        {
            state = targetState;
            triggerEnter = true;
        }

        public void LoadFromCSV(string filepath)
        {
            dialogs = new List<DialogData>();
            var path = Path.Combine(Application.streamingAssetsPath, filepath);
            var lines = File.ReadAllLines(path).Skip(1); // Skip header

            foreach (var line in lines) {
                var values = line.Split(',');
                var data = new DialogData
                {
                    id = int.Parse(values[0]),
                    characterName = values[1],
                    characterExpression = values[2],
                    characterPosition = values[3],
                    eventImage = values[4],
                    soundEffect = values[5],
                    dialogText = values[6],
                    choices = ParseChoices(values[7]),
                    nextSceneIDs = ParseNextSceneIds(values[8], values[0]),
                    animation = values[9],
                    voiceClip = values[10],
                    displayType = values[11]
                };
                dialogs.Add(data);
            }
            currentID = dialogs[0].id;  // Reset index when loading, ID#0 is reserved.
        }
```

```
103         public void Play() {
104             AVGUIManager.Instance.AVGUILoadDialog(GetCurrentDialog());
105             GoToState(STATE.RUNNING);
106         }
107
108         public void Stop(bool hide = true) {
109             if (hide) { AVGUIManager.Instance.AVGUIHide(); }
110             GoToState(STATE.FINISHED);
111         }
112
113         public void Pause(bool value) {
114             if (value) {
115                 GoToState(STATE.PAUSED);
116             }
117             else {
118                 GoToState(STATE.RUNNING);
119             }
120         }
121
122         private List<string> ParseChoices(string choicesStr)
123         {
124             if (string.IsNullOrEmpty(choicesStr) || choicesStr == "[]") {
125                 return new List<string>();
126             }
127             return choicesStr.Trim('[', ']').Split('|').ToList();
128         }
129
130         private List<int> ParseNextSceneIds(string nextSceneIDsStr, string currentSceneID)
131         {
132             if (string.IsNullOrEmpty(nextSceneIDsStr)) {
133                 return new List<int> { int.Parse(currentSceneID) + 1};
134             }
135             print(nextSceneIDsStr);
136             return nextSceneIDsStr.Split('|').Select(int.Parse).ToList();
137         }
```

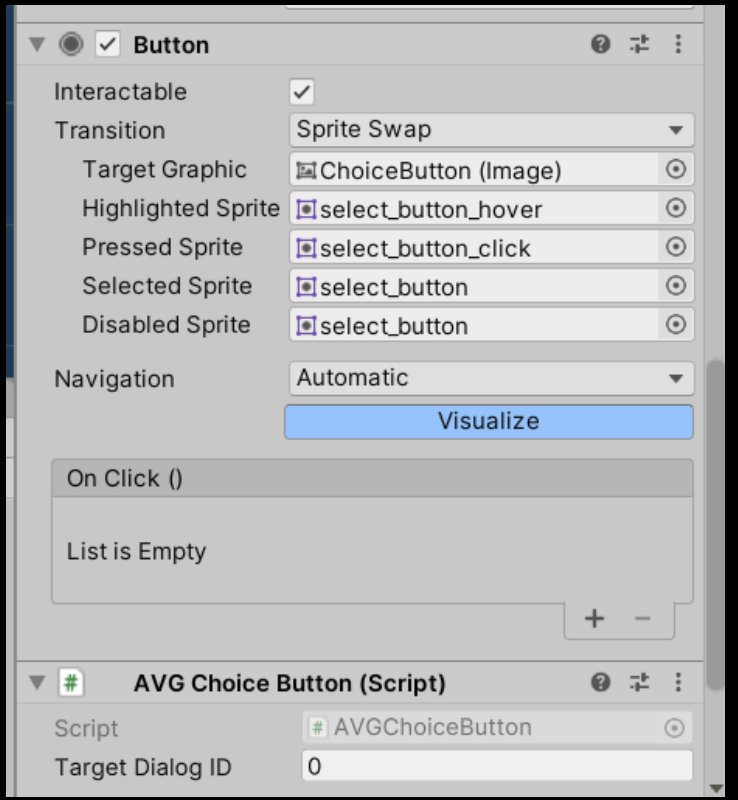
```
138
139         private DialogData GetCurrentDialog() {
140             return currentID > 0 ? GetDialogById(currentID) : null;
141         }
142
143         private DialogData GetNextDialog() {
144             DialogData nextDialog = GetDialogById(GetCurrentDialog().nextSceneIDs[0]);
145             return nextDialog;
146         }
147
148         private DialogData GetDialogById(int id) {
149             return dialogs?.Find(d => d.id == id);
150         }
151
152         internal void SetNextDialog(DialogData nextDialog)
153         {
154             if (nextDialog != null) {
155                 currentID = nextDialog.id;
156                 AVGUIManager.Instance.AVGUILoadDialog(nextDialog);
157             }
158             else {
159                 GoToState(STATE.FINISHED);
160             }
161         }
162
163         internal void SetNextDialog(int id){
164             DialogData nextDialog = GetDialogById(id);
165             SetNextDialog(nextDialog);
166         }
167
168         public bool IsFinshed()
169         {
170             return (state == STATE.FINISHED);
171         }
172     }
173 }
```

Add AVGChoiceButton On ChoiceButton Prefab

- Everytime we create a ChoiceButton into the scene, we hope to record info about which line is the next dialog line on it.
- By the way, we plan to write reponse action for the button in this new script.




```
1   using UnityEngine;
2
3   namespace AVG
4   {
5       public class AVGChoiceButton : MonoBehaviour
6       {
7           public int targetDialogID;
8
9           public void BranchDialog()
10          {
11              AVGMachine.Instance.SetNextDialog(targetDialogID);
12              AVGMachine.Instance.Pause(false);
13          }
14      }
15  }
```



Edit AVGUIPanel to Support Audio and ChoiceButton

- Add PlayAudio. All the audio files should be put in Resources/Audio folder.

```
80              for (int i = 0; i < choices.Count; i++) {
81                  GameObject button = Instantiate(buttonPrefab);
82                  button.transform.parent = choicePanel.transform;
83                  button.transform.localScale = Vector3.one;
84                  button.GetComponentInChildren<TextMeshProUGUI>().text = choices[i];
85                  var choiceButton = button.GetComponentInChildren<AVGChoiceButton>();
86                  choiceButton.targetDialogID = nextSceneIDs[i];
87                  button.GetComponentInChildren<Button>().onClick.AddListener( ()=> choiceButton.BranchDialog());
88              }
89          }
90
91          public void PlayAudio(string voiceClip) {
92              var voice = Resources.Load<AudioClip> ($"Audio/{voiceClip}");
93              if (voice != null) {
94                  var aSource = GetComponent<AudioSource>();
95                  if (aSource == null) {
96                      aSource = gameObject.AddComponent<AudioSource>();
97                  }
98                  aSource.Stop();
99                  aSource.clip = voice;
100                 aSource.Play();
101             }
102         }
103     }
104 }
```

Edit AVGUIManager to Support Audio

- Call PlayAudio in ”normal” and ”choice” dialogs. ❶❷
- Let AVGMachine go to PAUSE state under ”choice” dialog. ❷

```
29          switch (dialog.displayType) {
30              case "normal":
31                  characterImage = Resources.Load<Sprite>(
32                      $"Textures/Characters/{dialog.characterName}/{dialog.characterExpression}");
33                  dialogContent = dialog.dialogText;
34                  panel.ShowCharacter(characterImage, true, dialog.characterPosition);
35                  panel.ShowEvent(null, false);
36                  panel.ShowDialogBox(dialog.characterName, dialogContent);
37                  panel.ShowChoices(null, null, null, false);
38          ❶       panel.PlayAudio(dialog.voiceClip);
39                  break;
40              case "event":
41                  eventImage = Resources.Load<Sprite>(
42                      $"Textures/Events/{dialog.eventImage}");
43                  dialogContent = dialog.dialogText;
44                  panel.ShowCharacter(null, false);
45                  panel.ShowEvent(eventImage, true);
46                  panel.ShowDialogBox("", dialogContent);
47                  panel.ShowChoices(null, null, null, false);
48                  break;
49              case "choice":
50                  characterImage = Resources.Load<Sprite>(
51                      $"Textures/Characters/{dialog.characterName}/{dialog.characterExpression}");
52                  dialogContent = dialog.dialogText;
53                  panel.ShowCharacter(characterImage, true, dialog.characterPosition);
54                  panel.ShowEvent(null, false);
55                  panel.ShowDialogBox(dialog.characterName, dialogContent);
56                  panel.ShowChoices(dialog.choices, dialog.nextSceneIDs, choicePrefab, true);
57          ❷       panel.PlayAudio(dialog.voiceClip);
58                  AVGMachine.Instance.Pause(true);
59                  break;
60              case "narration":
61                  dialogContent = dialog.dialogText;
62                  panel.ShowCharacter(null, false);
63                  panel.ShowEvent(null, false);
64                  panel.ShowDialogBox("", dialogContent);
65                  panel.ShowChoices(null, null, null, false);
66                  break;
67              default:
68                  break;
```

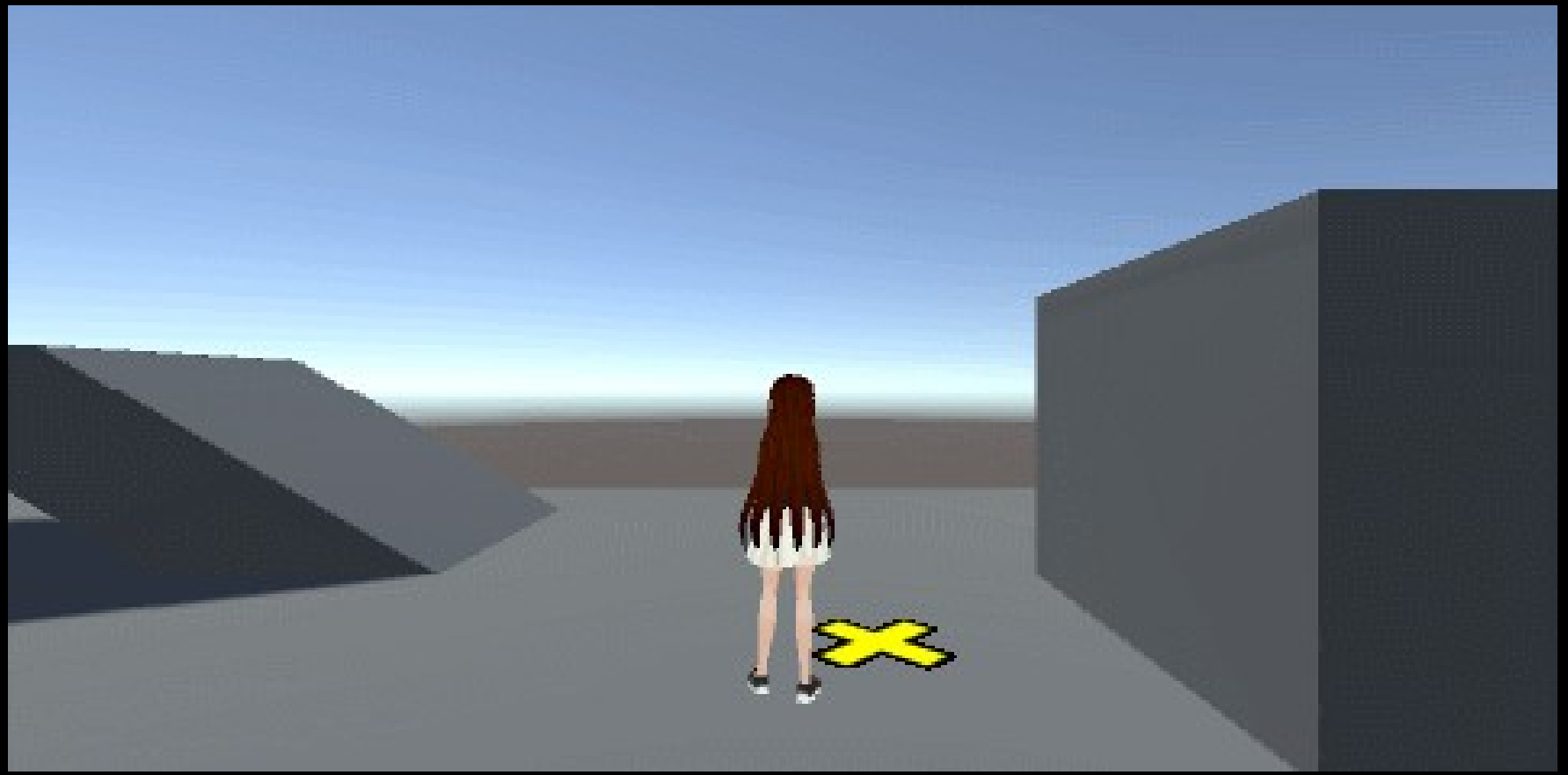
Test Client3

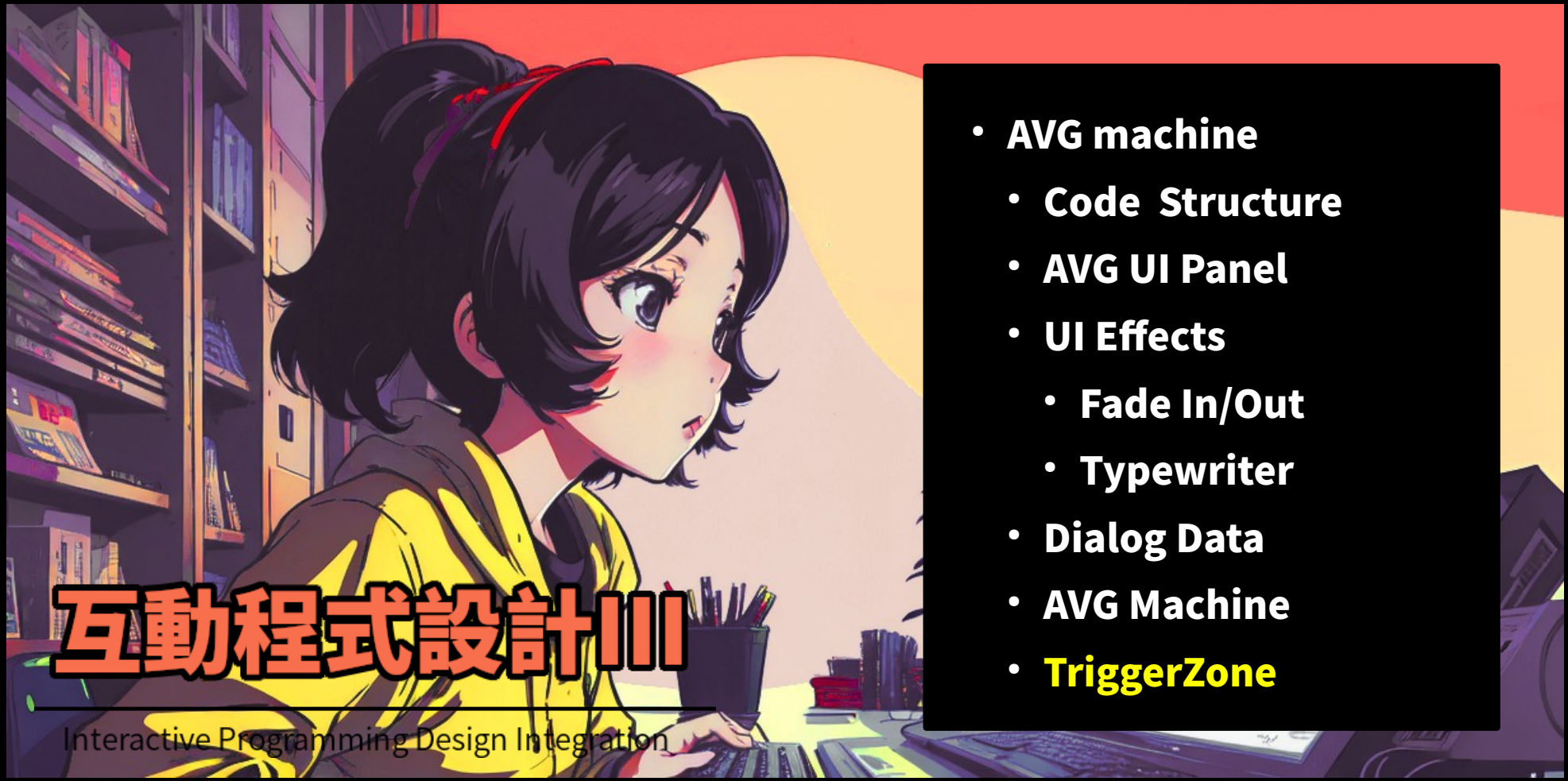
- Simplified Client Logic
  - This final implementation showcases a clean and concise block of code for the client logic.
  - The streamlined design is a result of all the efforts and setups completed during this lesson.
- Ease of Use
  - By organizing your CSV files and assets properly, you can easily create and manage new dialog scripts.
- Code Overview
  - Start Method:
    - Stops the AVGMachine to prepare for new dialog sequences.
  - Update Method:
    - Detects key press (1) to:
      - Load a dialog script (scene1.csv) using AVGMachine.Instance.LoadFromCSV().
      - Start playing the dialog using AVGMachine.Instance.Play().

```
1  using UnityEngine;
2  using AVG;
3
4  public class TestClient3 : MonoBehaviour
5  {
6      private void Start()
7      {
8          AVGMachine.Instance.Stop();
9      }
10
11     void Update()
12     {
13         if (Input.GetKeyDown("1"))
14         {
15             AVGMachine.Instance.LoadFromCSV("AVGScripts/scene1.csv");
16             AVGMachine.Instance.Play();
17         }
18     }
19 }
```

Test

- Test is. Press I to start dialog.





- AVG machine
  - Code Structure
  - AVG UI Panel
  - UI Effects
    - Fade In/Out
    - Typewriter
  - Dialog Data
  - AVG Machine
  - TriggerZone

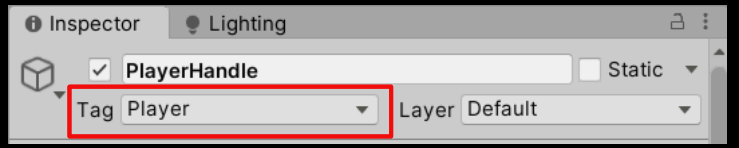
Trigger Zone

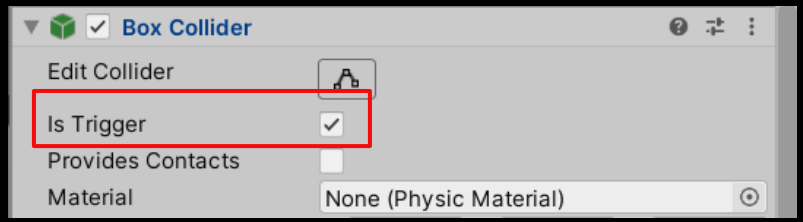
- Member Fields:
  - scriptPath: Path to the dialog script file.
  - isRetriggerable: Boolean flag to allow or prevent retriggering.
  - hasTriggered: Internal flag to track trigger activation.
- OnTriggerEnter Logic:
  - Detects if the Player enters the trigger zone.
  - Checks if the dialog has finished or the trigger is retriggerable.
  - If valid:
    - Marks the trigger as activated (hasTriggered = true).
    - Loads the dialog script from the specified scriptPath.
    - Starts the dialog sequence using AVGMachine.Instance.Play().

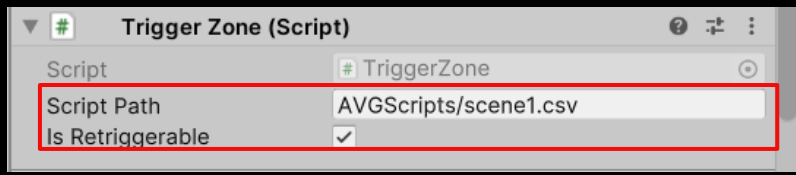
```
1  using UnityEngine;
2  using AVG;
3
4  public class TriggerZone : MonoBehaviour
5  {
6      [SerializeField] protected string scriptPath;
7      [SerializeField] private bool isRetriggerable = false;
8      private bool hasTriggered = false;
9
10     private void OnTriggerEnter(Collider other)
11     {
12         if (other.CompareTag("Player")
13             && AVGMachine.Instance.IsFinshed()
14             && (!hasTriggered || isRetriggerable))
15         {
16             hasTriggered = true;
17             AVGMachine.Instance.LoadFromCSV(scriptPath);
18             AVGMachine.Instance.Play();
19         }
20     }
21 }
22
```

Trigger Zone

- Setup the Scene
  - Create two Cubes in the scene and configure their colliders to act as triggers.
  - Add a new TriggerZone class and attach it to both cubes.
- Script Configuration
  - Set the scriptPath field to the respective CSV file paths under StreamingAssets.
    - Example: Assign one trigger for scene1.csv and the other for scene2.csv.
- Retrigger Options
  - Use the isRetriggerable field to control whether the trigger can be activated multiple times.
  - Ensure this option is configured appropriately for both triggers.







Mute Character When AVG is ON

- Modify the InputManager Class
  - Update the InputManager to disable character movement while the AVGMachine is running.
- Add a New Method
  - Implement a method called MuteCharacterMove(bool _isMute).
  - his method will control whether character movement is enabled or disabled.
- Integrate the Method
  - Call the MuteCharacterMove method at the beginning of the Update function.
  - ass the status of AVGMachine (whether it has finished running) to _isMute.

```
using UnityEngine.Events;
using UnityEngine;
using AVG;

public abstract class InputManager : MonoBehaviour
{
    public UnityEvent<Vector3> evtDpadAxis;
    public UnityEvent<bool> evtJump;
    public UnityEvent<bool> evtRun;
    public UnityEvent<bool> evtDialogClick;

    protected abstract void CalculateDpadAxis();
    protected abstract void CalculateJump();
    protected abstract void CalculateRun();
    protected abstract void CalculateDialogClick();
    protected abstract void PostProcessDpadAxis();
    protected abstract void MuteCharacterMove(bool _isMute);

    private void Update()
    {
        MuteCharacterMove(!AVGMachine.Instance.IsFinshed());
        CalculateDpadAxis();
        CalculateJump();
        CalculateRun();
        CalculateDialogClick();
        PostProcessDpadAxis();
    }
}
```

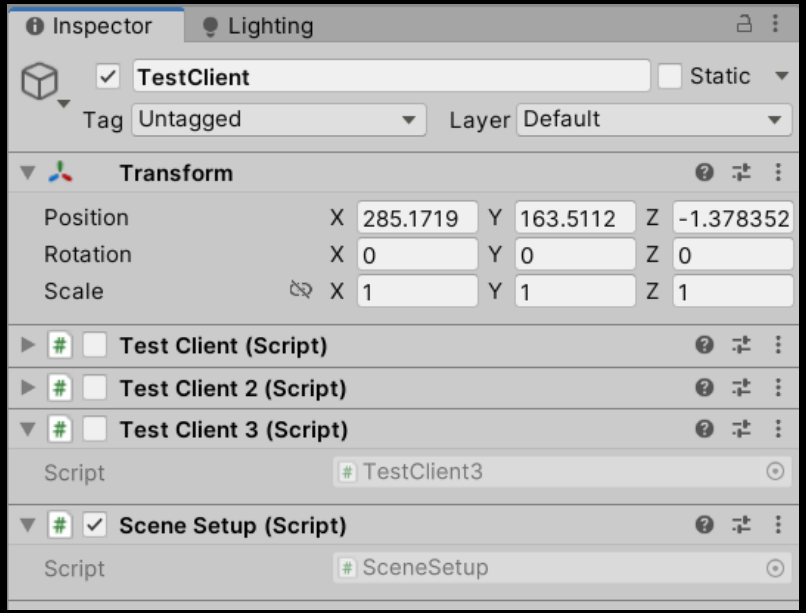
- Modify the KeyboardInputManager Class
  - Edit the KeyboardInputManager to incorporate logic for muting character movement.
- Use Ternary Operators
  - Leverage ternary operators to toggle arguments for character movement based on the mute status (isMute).
- Add the MuteCharacterMove Method
  - Implement the MuteCharacterMove(bool isMute) method to control the isMute variable and manage the movement state.
- Key Updates in Methods:
  - CalculateDpadAxis: Sets axis values based on key inputs (W, A, S, D). Uses isMute to disable movement by invoking a zero vector (Vector3.zero) when muted.
  - CalculateJump and CalculateRun: Uses ternary operators to send appropriate input states (true or false) depending on the mute status.
  - MuteCharacterMove: Updates the isMute state, enabling or disabling character controls as required.

```
10      protected override void CalculateDpadAxis()
11      {
12          axis = Vector3.zero;
13          if (Input.GetKey("w"))
14          {
15              axis.z = 1.0f;
16          }
17          if (Input.GetKey("s"))
18          {
19              axis.z = -1.0f;
20          }
21          if (Input.GetKey("d"))
22          {
23              axis.x = 1.0f;
24          }
25          if (Input.GetKey("a"))
26          {
27              axis.x = -1.0f;
28          }
29
30          evtDpadAxis?.Invoke(isMute? Vector3.zero: axis);
31      }
32
33      protected override void CalculateJump()
34      {
35          jump = Input.GetKeyDown("space");
36          evtJump?.Invoke(isMute? false: jump);
37      }
38
39      protected override void CalculateRun()
40      {
41          run = Input.GetKey("left shift");
42          evtRun?.Invoke(isMute? false: run);
43      }
44
45      protected override void CalculateDialogClick()
46      {
47          dialogClick = Input.GetKeyDown("mouse 0");
48          evtDialogClick?.Invoke(dialogClick);
49      }
50
51      protected override void PostProcessDpadAxis()...
55
56      protected override void MuteCharacterMove(bool _isMute)
57      {
58          isMute= _isMute;
59      }
60  }
61
```

Initialize AVGMachine

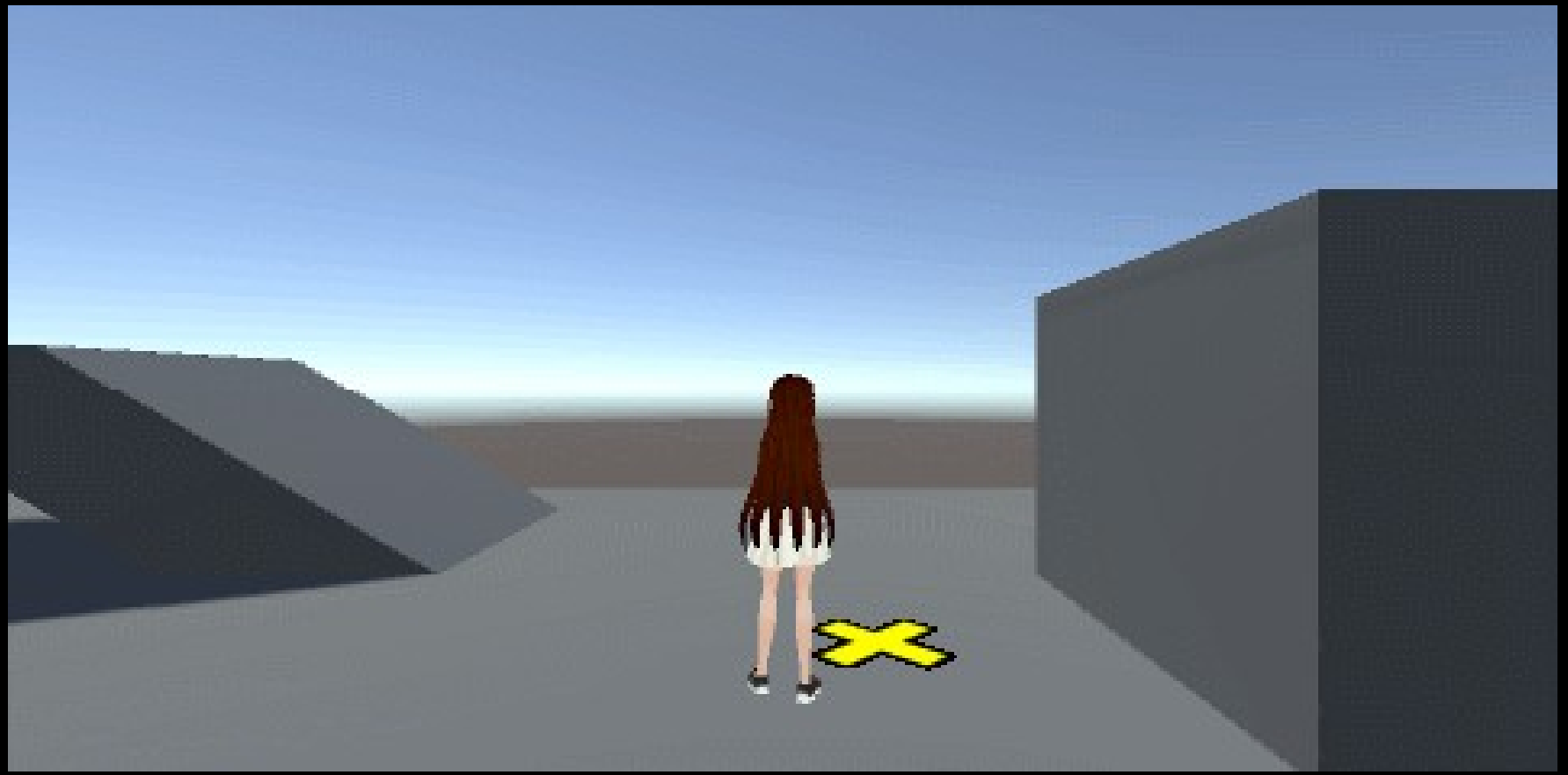
- Proper Initialization
  - To ensure the AVGMachine is initialized correctly, invoke the Stop() method before using it.
- Create a Setup Script
  - Define a new script called SceneSetup.cs.
  - Attach this script to the TestClient GameObject.
- Clean Up Test Logic
  - Remove all other TestClient logic to streamline the setup process.

```
1  using AVG;
2  using UnityEngine;
3
4  public class SceneSetup : MonoBehaviour
5  {
6      private void Start()
7      {
8          AVGMachine.Instance.Stop();
9      }
10 }
```



Test

- Test is. Press I to start dialog.



Future Optimizations

- Centralized String Management
  - Implement a static class to store and manage all literal strings used throughout the codebase, improving maintainability and reducing redundancy.
- Change Font Design
  - Support Chinese letters and make it stylized.
- Enforce Character Fade-In
  - Ensure characters fade in every time they reappear, even if the same character is speaking consecutively. This maintains a consistent visual flow.
- Advanced AVG Commands
  - Expand the dialog system to support additional commands, such as:
  - Camera Effects: Integrate camera shakes for dramatic events.
  - Cutscene Support: Trigger 3D animations or cutscenes within the AVG flow.
  - Scene Interaction: Allow dialogs to call methods on scene objects, enabling dynamic interactions.
- Audio Layering
  - Support multiple sound layers, such as background music, ambient sounds, and voice clips, with independent volume controls.
- Dynamic Character Expressions (like Live2D)
  - Link dialog lines to animated character expressions or poses to reflect emotions dynamically during conversations.


互動程式設計III
Interactive Programming Design Integration
Q&A