GAME OFF 2024

Pierre-César Bethelier: Music
Lino Martinez: Story – Writing - Direction
Brannigan Chateau: Lead Programming
Camille Péricat: Environment - Programming - Sound
Enzo Onesti: Game Design - Level Design - Programming
Ludovic Leroux: Game Design - Level Design
Nicolas Leroux: 3D Modeling

Existing data structures

- List
- Array

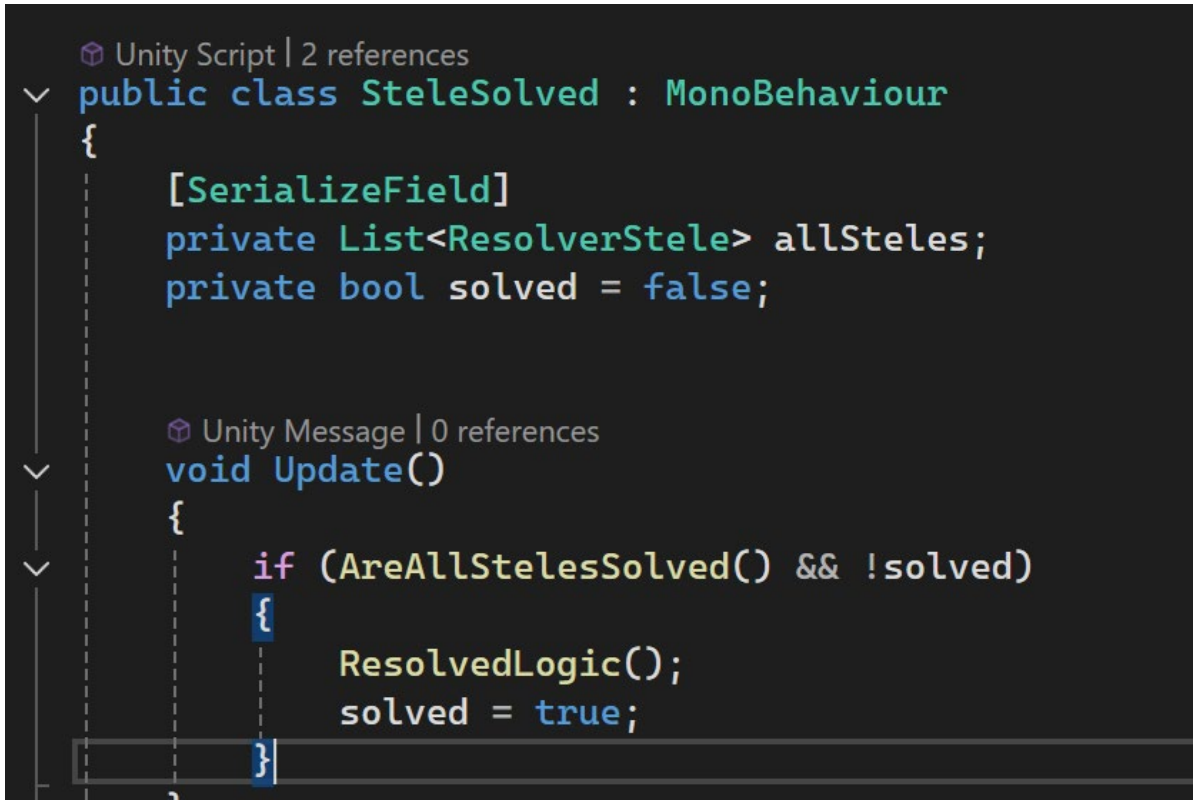
```
5
6   public class MainMenu : MonoBehaviour
7   {
8
9       [SerializeField] private GameObject cursor;
10
11      [SerializeField] private List<GameObject> menuItems = new List<GameObject>();
12      [SerializeField] private AudioSource audioSource;
13      [SerializeField] private AudioClip selectSound;
14      // Start is called once before the first execution of Update after the MonoBehaviour is created
15      void Start()
16      {
17
18      }
19
20      // Update is called once per frame
21      void Update()
22      {
23          foreach (var menuItem in menuItems)
24          {
25              Button button = menuItem.GetComponent<Button>();
26          }
27      }
```

```
public class TextOnInteract : MonoBehaviour
{
    [SerializeField] private Text messageText; // Référence au composant Text
    [SerializeField] private string[] messages; // Liste des messages
    [SerializeField] private float delayBetweenMessages; // Délai entre les messages
```

```
    private IEnumerator DisplayMessages()
    {
        foreach (string message in messages)
        {
            messageText.text = message; // Mettre à jour le texte
            Debug.Log(message);
            yield return new WaitForSeconds(delayBetweenMessages); // Attendre le délai
        }

        messageText.text = ""; // Nettoyer le texte après la fin des messages
    }
}
```

List and Array for the puzzle logic (symbols combination)

```
y-CSharp                                    ButtonStele                    is
 9
10
11      [SerializeField] private GameObject[] highlight;
12      [SerializeField] private LayerMask interactible;
13      private bool hover = false;
14
        Unity Message | 0 references
15      private void Update()
16      {
17          if (hover == false)
18          {
19              foreach (GameObject obj in highlight)
20              {
21                  obj.SetActive(false);
22              }
23          }
24          else
25          {
26              hover = false;
27          }
28      }
```



```
Unity Script | 2 references
public class SteleSolved : MonoBehaviour
{
    [SerializeField]
    private List<ResolverStele> allSteles;
    private bool solved = false;


    Unity Message | 0 references
    void Update()
    {
        if (AreAllStelesSolved() && !solved)
        {
            ResolvedLogic();
            solved = true;
        }
```

Replace a list by an array

- We have a limited number of item slots, so it is more efficient to use an array

```
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.UI;
using static UnityEngine.Rendering.DebugUI;

public class Inventory : MonoBehaviour
{
    [Header("Inventory Settings")]
    public List<PickupController> pickedUpItems = new List<PickupController>();
    public static int itemSlot = 7;
    public Pickup[] pickedUpItems;
    public Camera playerCamera;
    public KeyCode interactKey = KeyCode.E;
    public KeyCode pickKey = KeyCode.E;
    public float interactRange = 5f;
    public float pickupRange = 3f;

    private void Update()
    private IInteractable currentHoverItem;

    private void Awake()
    {
        HandlePickup();
        HandleInteraction();
        pickedUpItems = new Pickup[itemSlot];
    }
```

```
Unity Script (1 asset reference) | 0 references
public class Inventory : MonoBehaviour
{
    [Header("Inventory Settings")]
    public static int itemSlot = 7;
    public Pickup[] pickedUpItems;
    public Camera playerCamera;
    public KeyCode interactKey = KeyCode.E;
    public KeyCode pickKey = KeyCode.E;
    public float interactRange = 5f;
    public float pickupRange = 3f;

    private IInteractable currentHoverItem;

    Unity Message | 0 references
    private void Awake()
    {
        pickedUpItems = new Pickup[itemSlot];
    }
```

Linked list implementation

- Create a linked list instead of a list

```
Unity Script (5 asset references) | 2 references
public class ResolverStele : MonoBehaviour
{
    public List<ButtonStele> buttonSteleList;
    public LinkedList<ButtonStele> buttonSteleLinkedList = new LinkedList<ButtonStele>();

    [SerializeField] bool isSphere = false;

    public bool hasCheckedCorrect;

    [SerializeField]
    private float timeToCheck = .5f;
    private float elapsedTime = 0f;

    [SerializeField]
    private AudioSource errorAudio;

    Unity Message | 0 references
    private void Awake()
    {
        if (buttonSteleList != null)
        {
            buttonSteleLinkedList = new LinkedList<ButtonStele>(buttonSteleList);
        }
```

```
LinkedListNode<ButtonStele> button = buttonSteleLinkedList.First;

while (button != null)
{
    if (button.Value.isGood)
        goodCount++;
    if (button.Value.isGood && button.Value.isEnabled)
        enabledGoodCount++;
    if (button.Value.isEnabled)
        enabledCount++;

    button = button.Next;
}

if (enabledGoodCount == goodCount && enabledCount == enabledGoodCount)
{
    Debug.Log("Correct");
    LinkedListNode<ButtonStele> buttonToCheck = buttonSteleLinkedList.First;
    while (buttonToCheck != null)
    {
        buttonToCheck.Value.solved = true;
        buttonToCheck = buttonToCheck.Next;
    }
```

Implementation of a new algorithm : Dynamic Programming

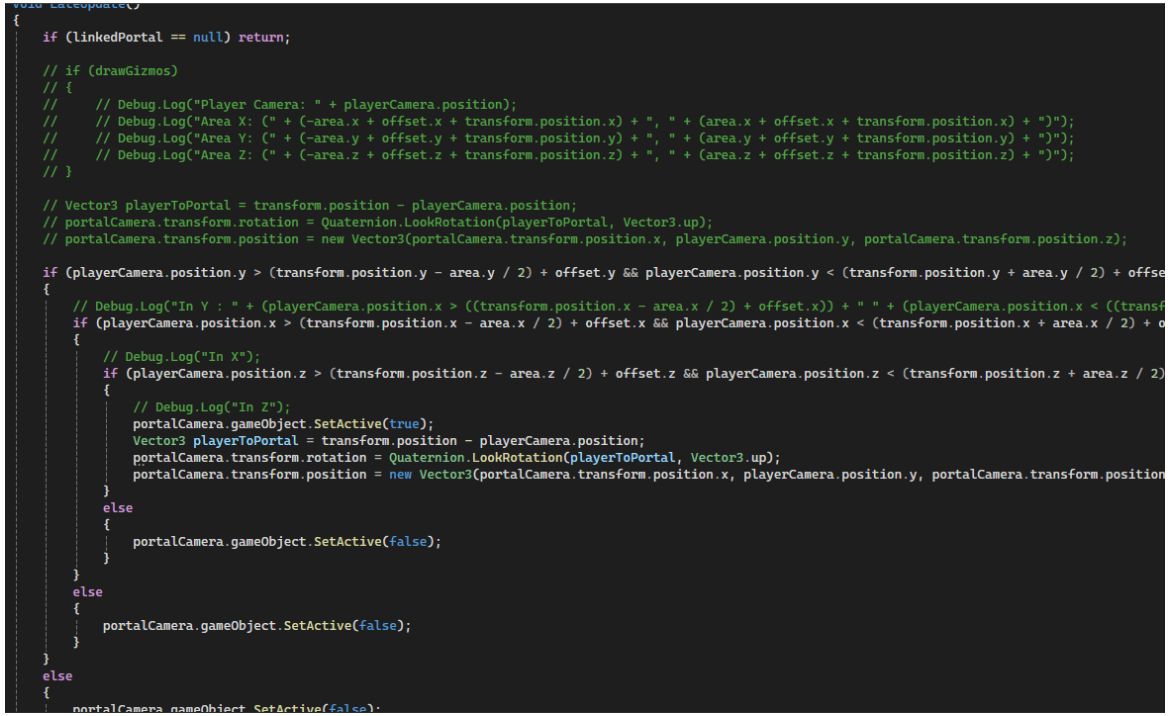
Dynamic programming is a set of methods for solving optimization problems efficiently by breaking them down into smaller subproblems

```
-1,28 +1,1 @@
-using UnityEngine;
-using UnityEngine.UI;

-public class PickupController : MonoBehaviour
-{
-    [Header("References")]
-    public Inventory inventory;
-    private void PickUp()
-    {
-        inventory.AddItem(this);
-        gameObject.SetActive(false);
-    }

-    public void AttemptPickup(Camera playerCamera, float pickupRange)
-    {
-        if (UserInput.Instance.InteractInput)
-        {
-            Ray ray = playerCamera.ScreenPointToRay(Input.mousePosition);
-            if (Physics.Raycast(ray, out RaycastHit hit, pickupRange))
-            {
-                if (hit.collider.gameObject == gameObject)
-                {
-                    PickUp();
-                }
-            }
-        }
-    }
-}
+
```

```
    1 reference
    private void AttemptInteraction(IInteractable interactable)
    {
        foreach (var item in pickedUpItems)
        {
            if (interactable.CanInteract(item))
            {
                interactable.Interact();
            }
        }
    }

    1 reference
    private void PickUp(Pickup item)
    {
        AddItem(item);
        item.gameObject.SetActive(false);
    }
}
```

Code refactoring

- Too long IF statements
- Unnecessary/redundant ELSE statements



```
void LateUpdate()
{
    if (linkedPortal == null) return;

    // if (drawGizmos)
    // {
    //     // Debug.Log("Player Camera: " + playerCamera.position);
    //     // Debug.Log("Area X: (" + (-area.x + offset.x + transform.position.x) + ", " + (area.x + offset.x + transform.position.x) + ")");
    //     // Debug.Log("Area Y: (" + (-area.y + offset.y + transform.position.y) + ", " + (area.y + offset.y + transform.position.y) + ")");
    //     // Debug.Log("Area Z: (" + (-area.z + offset.z + transform.position.z) + ", " + (area.z + offset.z + transform.position.z) + ")");
    // }

    // Vector3 playerToPortal = transform.position - playerCamera.position;
    // portalCamera.transform.rotation = Quaternion.LookRotation(playerToPortal, Vector3.up);
    // portalCamera.transform.position = new Vector3(portalCamera.transform.position.x, playerCamera.position.y, portalCamera.transform.position.z);

    if (playerCamera.position.y > (transform.position.y - area.y / 2) + offset.y && playerCamera.position.y < (transform.position.y + area.y / 2) + offse
    {
        // Debug.Log("In Y : " + (playerCamera.position.x > ((transform.position.x - area.x / 2) + offset.x)) + " " + (playerCamera.position.x < ((transf
        if (playerCamera.position.x > (transform.position.x - area.x / 2) + offset.x && playerCamera.position.x < (transform.position.x + area.x / 2) + o
        {
            // Debug.Log("In X");
            if (playerCamera.position.z > (transform.position.z - area.z / 2) + offset.z && playerCamera.position.z < (transform.position.z + area.z / 2)
            {
                // Debug.Log("In Z");
                portalCamera.gameObject.SetActive(true);
                Vector3 playerToPortal = transform.position - playerCamera.position;
                portalCamera.transform.rotation = Quaternion.LookRotation(playerToPortal, Vector3.up);
                portalCamera.transform.position = new Vector3(portalCamera.transform.position.x, playerCamera.position.y, portalCamera.transform.position
            }
            else
            {
                portalCamera.gameObject.SetActive(false);
            }
        }
        else
        {
            portalCamera.gameObject.SetActive(false);
        }
    }
    else
    {
        portalCamera.gameObject.SetActive(false);
```

```
float playerCamY = playerCamera.position.y;
float portalOffsetYMinus = (transform.position.y - area.y / 2) + offset.y;
float portalOffsetYPlus = (transform.position.y + area.y / 2) + offset.y;
float playerCamX = playerCamera.position.x;
float portalOffsetXMinus = (transform.position.x - area.x / 2) + offset.x;
float portalOffsetXPlus = (transform.position.x + area.x / 2) + offset.x;
float playerCamZ = playerCamera.position.z;
float portalOffsetZMinus = (transform.position.z - area.z / 2) + offset.z;
float portalOffsetZPlus = (transform.position.z + area.z / 2) + offset.z;

float portalCamZ = portalCamera.transform.position.z;
float portalCamX = portalCamera.transform.position.x;

portalCamera.gameObject.SetActive(false);

if (playerCamY > portalOffsetYMinus && playerCamY < portalOffsetYPlus)
{
    if (playerCamX > portalOffsetXMinus && playerCamX < portalOffsetXPlus)
    {
        if (playerCamZ > portalOffsetZMinus && playerCamZ < portalOffsetZPlus)
        {
            portalCamera.gameObject.SetActive(true);
            Vector3 playerToPortal = transform.position - playerCamera.position;
            portalCamera.transform.rotation = Quaternion.LookRotation(playerToPortal, Vector3.up);
            portalCamera.transform.position = new Vector3(portalCamX, playerCamY, portalCamZ);
        }
    }
}
```

Accessibility improvement and bug fix

- Player Movement
- Interactable Books

```
29          }
30
            2 references
31          public override void Interact()
32          {
33              isInteracting = !isInteracting;
34              foreach (GameObject obj in toDisable)
35              {
36                  obj.SetActive(false);
37              }
38
39              canvas.SetActive(isInteracting);
40              page.SetActive(isInteracting);
41          }
42      }
43
```

```
void Update()
{
    grounded = Physics.Raycast(transform.position, Vector3.down, playerHeight * 0.7f + 0.
    MyInput();
    StateHandler();
    if (grounded)
        rb.linearDamping = groundDrag;
    else
        rb.linearDamping = 10;
}
```