Projet Final

Par Marvin Tanonkou

Dans ce document, je vais vous présenter les structures de données utiliser dans le gameplay dans le cadre de mon cours en structure de données.

J'ai travaillé sur la visualisation des objets dans l'inventaire du joueur dans mon jeu Murder Mine. Il permet au joueur de voir les objects qu'il a présentment dans son inventaire.

Structure De Donnée : Listes

Le script ***ItemDisplayer*** contient une ***liste de Transforms et d'Items*** qui jouent des rôles majeurs dans la représentation visuelle de l'inventaire.

La **liste de transform** me **permet de changer la position de l'objet** et la liste des Items les caractéristiques d'un item que le joueur visualise en utilisant une variable de type interger qui agit comme un index qui s'appelle currentIndex.

```
private List<Transform> itemDisplayers;
private Item[] items = null;
private int currentIndex = 0;
```

La fonction DisplayItems gere le visuel des Items. Elle est appelée à chaque frame du jeu

```
Unity Message | 0 references
private void LateUpdate()
{
    if (!canUpdate) return;
    DisplayItems(items.ToArray(), itemDisplayers.ToArray());
}
```

Elle gere certains cas spécifiques si l'inventaire est nul ou qu'il n'y a pas d'éléments et ainsi mets les éléments de UI et les positions des Items à jours.

```
1 reference
private void DisplayItems(Item[] items, Transform[] transforms)
{
    if (items == null) return;

    if(items.Length > 0)
    {
        Item currentItem = items[currentIndex];

        currentItem.GainExperience(10f * Time.deltaTime);
        currentItem.UpdateInput();

        SetLevelupFillBar (currentItem);
        SetInformationText(currentItem);
        SetAbilityText    (currentItem);

        DisplayItem(transforms, currentIndex, Vector3.up * currentItemPositionY);
        DisplayItems(transforms, itemDisplayers[currentIndex]);
    }
}
```

S’il y a plus que 1 Transform dans la liste des Transforms, elle va positionner les Items de gauche à droite selon l’index de l’Item choisi. Donc si l’ index plus que celui de l’item choisi, elle va mettre l’Item vers la gauche, sinon c’est l’inverse.

```
1 reference
private void DisplayItems(Transform[] transforms, Transform currentTransform)
{
    bool isOppositeDisplay = false;
    List<Transform> leftSideItems;
    List<Transform> rightSideItems;

    if (transforms == null) return;
    else if (transforms.Length <= 1) return;

        leftSideItems = new List<Transform>();
    rightSideItems = new List<Transform>();

    foreach (Transform transform in transforms)
    {
        if (transform == currentTransform)
        {
            isOppositeDisplay = !isOppositeDisplay;
            continue;
        }

        if (!isOppositeDisplay)
            leftSideItems.Add(transform);
        else
            rightSideItems.Add(transform);
    }

    leftSideItems.Reverse();

    Display(leftSideItems.ToArray(),  Vector3.left  * itemSpacing);
    Display(rightSideItems.ToArray(), Vector3.right * itemSpacing);

    void Display(Transform[] transforms, Vector3 desiredPosition)
    {
        for (int i = 0; i < transforms.Length; i++)
            DisplayItem(transforms[i], desiredPosition * (i + 1));
    }

}
```

La Fonction DisplayItem mets l'Item sélectionné visible devant les yeux du joueur.

```
1 reference
private void DisplayItem(Transform[] transforms, int selectableIndex, Vector3 desiredPosition)
{
    ClampItemIndex(items, ref selectableIndex);
    DisplayItem(transforms[selectableIndex], desiredPosition);
}

2 references
private void DisplayItem(Transform transform, Vector3 desiredPosition)
{
    if (transform == null) return;
    transform.localEulerAngles += Time.deltaTime * Vector3.up * rotationSpeed;
    transform.position = Vector3.Lerp(transform.position, desiredPosition, Time.deltaTime * positionLerpSpeed);
}
```

Il y a une fonction qui s'appelle **SortArray** qui utilise le bubble sort sur la liste des Tranforms et des Items afin de permettre dès les ordonnés tout dépendamment de l'identifiant de l'Item; En ordre croissant avec la variable ID de l'Item.

```
1 reference
private async Task<Item[]>SortArray(Item[] items)
{
    canUpdate = false;

    for (int i = 0; i < items.Length - 1; i++)
        for (int j = 0; j < items.Length - i - 1; j++)
            if (items[j].ID > items[j + 1].ID)
            {
                Item tempVar = items[j];
                items[j] = items[j + 1];
                items[j + 1] = tempVar;

                Transform tempTrans = itemDisplayers[j];
                itemDisplayers[j] = itemDisplayers[j + 1];
                itemDisplayers[j + 1] = tempTrans;

                VisualizeSorting(itemDisplayers[j], tempTrans, sortingSpeed);
                await Awaitable.WaitForSecondsAsync(sortingSpeed);
            }

    canUpdate = true;
    return items;
}
```

La fonction Visualize permet de visualiser les échanges de positions entre les Items qui font un sort.

```
async Task Visualize(Vector3 endPosition01, Vector3 endPosition02, float duration)
{
    float t = 0f;
    Vector3 startPosition01, startPosition02;

    startPosition01 = item01.transform.position;
    startPosition02 = item02.transform.position;

    while (t < duration)
    {
        float progress = t / duration;
        LerpPosition(item01.transform, startPosition01, endPosition01, progress);
        LerpPosition(item02.transform, startPosition02, endPosition02, progress);

        t += Time.deltaTime;
        await Task.Yield();
    }

    item01.transform.position = endPosition01;
    item02.transform.position = endPosition02;

    void LerpPosition(Transform transform, Vector3 startPosition, Vector3 endPosition, float progression)...
}
```

Structure De Donnée : Linked List

Chaque Items peuvent avoir des habilités uniques grâce à la classe ItemAbility. De façons progressive et tout dépendement de son niveau, l'Item peut débloquer plusieurs abilités.

```
[System.Serializable]
38 references
public class ItemAbility
{
    private string abilityName;

    private string abilityDescription;

    private uint unlockedLevel;

    private UnityEvent onActivate;
    public ItemAbility next;

    0 references
    public string AbilityName => abilityName;
    0 references
    public string AbilityDescription => abilityDescription;
    5 references
    public uint UnlockedLevel => unlockedLevel;


    1 reference
    public ItemAbility(string abilityName, string abilityDescription, uint unlockedLevel)
    {
        this.abilityName = abilityName;
        this.abilityDescription = abilityDescription;
        this.unlockedLevel = unlockedLevel;
    }

    1 reference
    public ItemAbility(string abilityName, string abilityDescription, uint unlockedLevel, UnityEvent onActivate) : this(abilityName, abilityDescription, unlockedLevel)
    {
        this.onActivate = onActivate;
    }

    1 reference
    public void Activate()
    {
        if (onActivate != null) onActivate.Invoke();
    }
    0 references
    public void AddOnActivateEvent(UnityAction onActivate)
    {
        if (onActivate != null && this.onActivate != null) this.onActivate.AddListener(onActivate);
    }
    0 references
    public void RemoveOnActivateEvent(UnityAction onActivate)
    {
        if (onActivate != null && this.onActivate != null) this.onActivate.RemoveListener(onActivate);
    }


    public string UnlockRequirement() => $"Requires <color=yellow>Level {unlockedLevel}</color> To Unlock Next Ability (Unlocks {abilityName})";

    public override string ToString() => $"Ability Name: {abilityName} | Description: {abilityDescription} | Unlock Level: {unlockedLevel}";

}
```

Par exemple, s'il débloque une habilité au niveau 1, le jeu va chercher avec la fonction FindNextAbility et LevelUp dans le script Inventory.

```
1 reference
private async Task<ItemAbility> FindNextAbility(ItemAbility mainAbility)
{
    ItemAbility current = mainAbility;

    await Task.Run(async () =>
    {
        bool foundNextAbility = false;

        while (current != null)
        {
            if (currentLevel < current.UnlockedLevel)
            {
                foundNextAbility = true;
                break;
            }

            current = current.next;
            await Task.Yield();
        }

        if (!foundNextAbility) { current = null; }
    });

    return current;
}
```

Augmente le niveau de l'Item.

```
1 reference
private void LevelUp()
{
    exprienceNeededToLevelup *= 2f;
    currentExperience = 0f;
    currentLevel++;
}
```

Au début du jeu, la linked list va être ordonné avec l'algorithme MergeSort pour faire que les abilités puisse être débloquer en ordre selon la variable unlockedLevel.

```
1 reference
private ItemAbility Split(ItemAbility head)
{
    ItemAbility result01 = head;
    ItemAbility result02 = head;

    while (result01 != null && result01.next != null)
    {
        result01 = result01.next.next;
        if (result01 != null)
        {
            result02 = result02.next;
        }
    }

    ItemAbility temp = result02.next;
    result02.next = null;
    return temp;
}
```

```
3 references
private ItemAbility Merge(ItemAbility ability01, ItemAbility ability02)
{
    if (ability01 == null) return ability02;
    if (ability02 == null) return ability01;

    if (ability01.UnlockedLevel < ability02.UnlockedLevel)
    {
        ability01.next = Merge(ability01.next, ability02);
        return ability01;
    }
    else
    {
        ability02.next = Merge(ability01, ability02.next);
        return ability02;
    }
}
```

```
3 references
private ItemAbility MergeSort(ItemAbility main)
{
    if (main == null || main.next == null)
        return main;

    ItemAbility other = Split(main);

    main = MergeSort(main);
    other = MergeSort(other);

    return Merge(main, other);
}
```

Difficultés Vécues

Pour être honnête, je n’ai pas trop vécu de difficulté durant le temps qui a été accorder pour le projet. Le seul problème qui m’a menacé était pour trouver une implémentation utile d’un Graph ou d’un Arbre. Je n’avais pas d’idée en tête qui pourrait être utiliser dans un tel contexte. J’aimerais bien en apprendre davantage comment les utiliser de façons efficaces dans mes prochains travaux scolaires ou personnelle.

Controls :
[TAB] = Inventaire.
[1-5] = Quick Slots
[WASD] = Bouger
[Mouse01] = Interagir