

Necrospire





Chapters

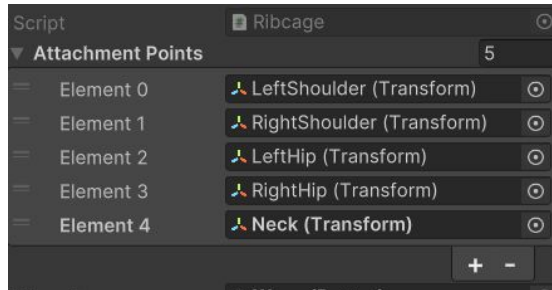
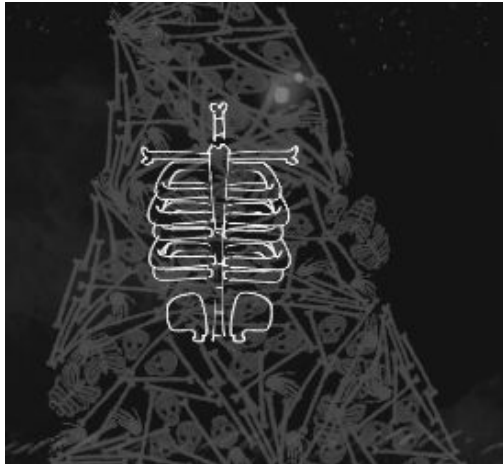
- Game development choices
- Task implementations and details
- Feedback Integration

The reason I chose to program a skeleton building game

- One of the main reasons was that I wanted to try something new. Every time I start a game project, I have always stuck to platformers because I was familiar with those, or any sort of action game. Making something else entirely proved to be interesting to code and it helps understanding other parts of game making that you would otherwise not get from making the same type of project over and over. Which should help with developing games in general, as you unlock new ways of thinking about other types of games.
- The skeleton part was at first supposed to be a full on monster, but that concept proved too much for coding something like this for the first time. And because we wanted to go for a spooky aesthetic anyway, the skeleton fit perfectly

Array

	Added BoneSpawner BoneSpawner prefab + organized prefabs
	Added Icons and Folders Human bone icons, Human bone prefabs, Prefab folder, Black Material
	Beginnings of the Main Menu
	Added Game Scene



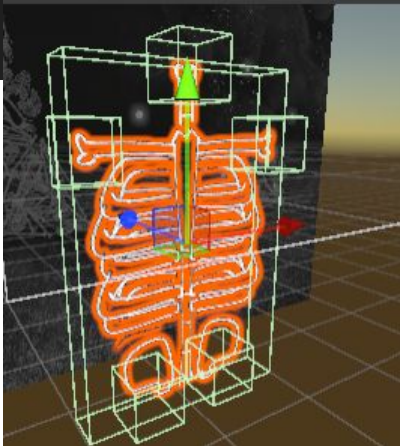
Uses of the array

Used for getting socket position

```
public Transform GetClosestSocket(Vector3 position)
{
    Transform closestSocket = null;
    float closestDistance = Mathf.Infinity;

    foreach (Transform socket in attachmentPoints)
    {
        float distance = Vector3.Distance(position, socket.position);
        if (distance < closestDistance)
        {
            closestSocket = socket;
            closestDistance = distance;
        }
    }

    return closestSocket;
}
```

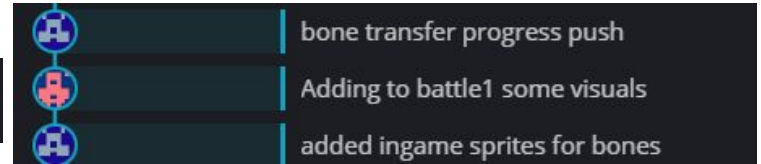


Used for wiggling animation

```
foreach (Transform socket in attachmentPoints)
{
    if (socket != null)
    {
        float randomRotation = Mathf.Sin(Time.time * wiggleSpeed) * currentBoneIntensity;
        socket.localRotation = Quaternion.Euler(0f, 0f, randomRotation);
    }
}
```



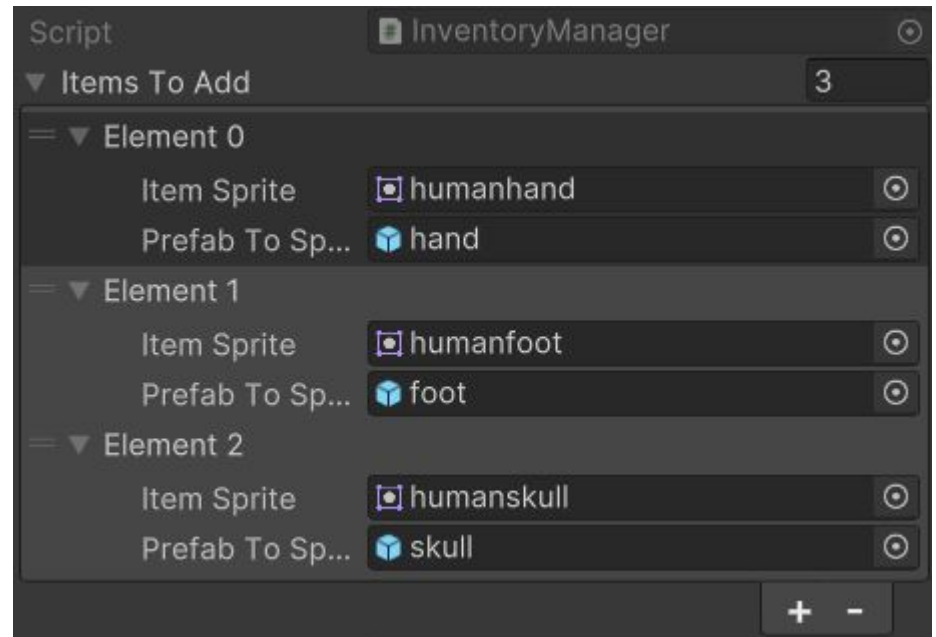
Linked List



```
[System.Serializable]
public class ItemData
{
    public Sprite itemSprite;
    public GameObject prefabToSpawn;

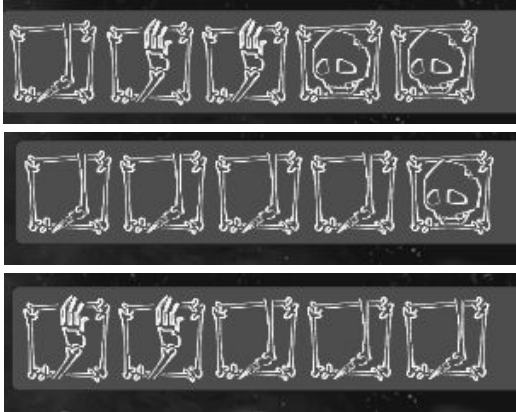
    [HideInInspector]
    public GameObject uiObject; // created at runtime
}

public List<ItemData> itemsToAdd = new List<ItemData>();
```



Uses of the Linked List

Randomize Inventory

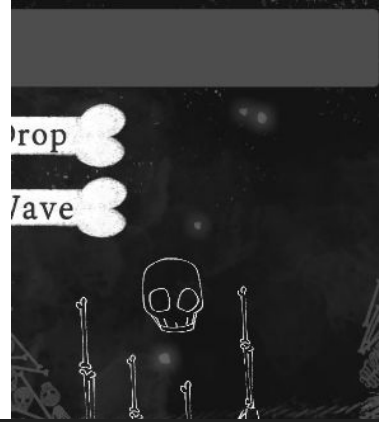


```
if (itemsToAdd.Count > 0)
{
    for (int i = 0; i < randomItemsToAdd; i++)
    {
        ItemData randomItem = itemsToAdd[Random.Range(0, itemsToAdd.Count)];

        // copy so u dont modify original
        ItemData newItem = new ItemData();
        newItem.prefabToSpawn = randomItem.prefabToSpawn;
        newItem.itemSprite = randomItem.itemSprite;

        AddItem(newItem);
    }
}
```

Drop items from inventory



```
IEnumerator DropItemsOneByOne()
{
    isDropping = true;

    while (inventoryItems.Count > 0)
    {
        ItemData item = inventoryItems.First.Value;
        inventoryItems.RemoveFirst();

        if (item.uiObject != null)
            Destroy(item.uiObject);

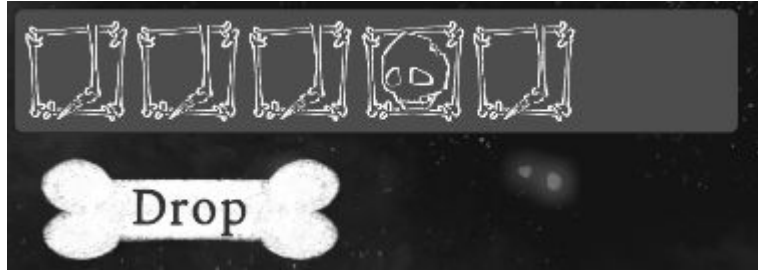
        if (item.prefabToSpawn != null)
            Instantiate(item.prefabToSpawn, spawnPoint.position, Quaternion.identity);

        yield return new WaitForSeconds(dropInterval);
    }

    isDropping = false;
}
```


FEEDBACK INTEGRATION

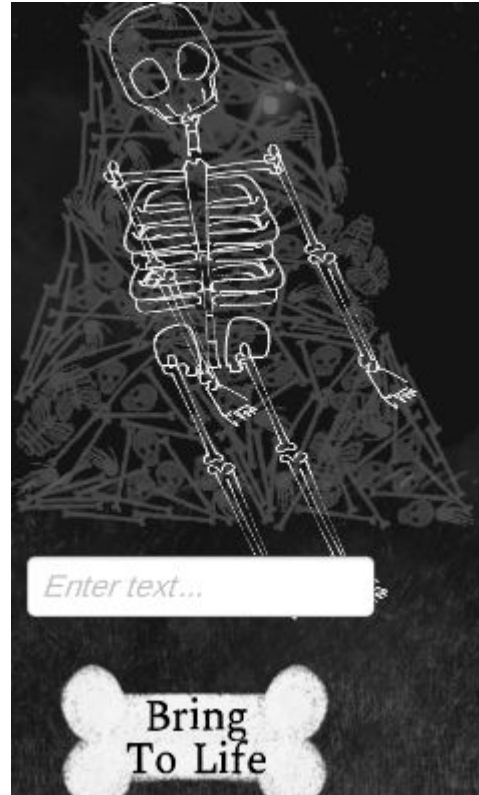
inventory physically drops



waving interaction



animated skeleton



naming skeleton

