Github Code Link

Bug 1: Order of contained objects is not maintained when saved to vault file nor when read from vault file.

Goal: Ensure that the VaultFile's "Objects" list maintains the same order of objects as the one found inside SaveableGunCase's m_containedObjects.

Why: SaveableGunCase assumes that the position and rotation of the object at index "i" of m_containedObjects can be found at index "i" of both m_containedPositions and m_containedRotations. We discovered below that while saving the VaultFile, the order of m_containedObjects is not maintained, and while spawning the vault file it's further shuffled and not maintained. This causes an issue where, with a likelihood correlated to the amount of unique objects inside a case, the objects will spawn in positions they don't belong to.

How:

1. Modified lines of **WriteToVaultFileObject**
    a. Line 16: Create a temporary objects array "MYobjs"
    b. Acknowledge that GetContainedObjectsRecursively maintains the order of m_containedObjects
    c. Line 17: Iterate over all objects scanned "objs"
        i. Line 19: Extract all its contained objects with GetContainedObjectsRecursively
        ii. Line 20: Add every scanned object to "MYobjs"
        iii. Line 21: Iterate over ordered contained objects
            1. Line 25: If contained object was added previously to "MYobjs" list, remove it from its earlier position
            2. Line 28: Add contained object to the back of the list
    d. Line 31: Set "objs" to "MYobjs" so all code below works the same as before

At the end of this code, we can guarantee that contained objects will appear After their parent object, and in the order in which they existed in m_containedObjects.

```
5     private static void WriteToVaultFileObject(List<FVRPhysicalObject> objs, VaultFile f, bool savePositionsRelativeToSpa
10          f.QuickbeltLayoutName = ManagerSingleton<GM>.Instance.QuickbeltConfigurations[GM.Options.QuickbeltOptions.Quick
11      }
12      Dictionary<FVRPhysicalObject, int> dictionary = new Dictionary<FVRPhysicalObject, int>();
13      Dictionary<FVRPhysicalObject, int> dictionary2 = new Dictionary<FVRPhysicalObject, int>();
14      Dictionary<FVRPhysicalObject, int> dictionary3 = new Dictionary<FVRPhysicalObject, int>();
15      Dictionary<FVRPhysicalObject, int> dictionary4 = new Dictionary<FVRPhysicalObject, int>();
16      List<FVRPhysicalObject> MYobjs = new List<FVRPhysicalObject>(); // MYobjs will replace objs after this loop.
17      for (int i = 0; i < objs.Count; i++)
18      {
19          List<FVRPhysicalObject> containedObjectsRecursively = objs[i].GetContainedObjectsRecursively();
20          MYobjs.Add(objs[i]); // Add parent container first.
21          for (int j = 0; j < containedObjectsRecursively.Count; j++)
22          {
23              // If contained object already added, remove it from its previous position,
24              // then add it after its parent.
25              if (MYobjs.Contains(containedObjectsRecursively[j])) {
26                  MYobjs.Remove(containedObjectsRecursively[j]);
27              }
28              MYobjs.Add(containedObjectsRecursively[j]);
29          }
30      }
31      objs = MYobjs; // objs is now MYobjs, which has all contained objects occurring after their parent container.
32      for (int k = 0; k < objs.Count; k++)
```

2. Modified lines of **SpawnVaultFileRoutine**
    a. Line 159: Change Dictionary<> to OrderedDictionary

Originally dicObjectsContainedInIndex was a Dictionary, a data structure which cannot keep the order of items added to it, so in line 362 the “foreach” statement could give us a shuffled order of items in the Vault file. By changing it to OrderedDictionary, we can rely on the “foreach” to give us the same order of items added in line 336, where “obj2” originates from a linear iteration over the vault file’s “f.Objects”, which we ensured in step 1 was ordered correctly.
At the end of this code, we can guarantee that the method “ContainOtherObject” is called in the same order as the original item order of the container object’s m_containedObjects.

Code | Blame | 403 lines (399 loc) · 19.2 KB | Code 55% faster with GitHub Copilot | Raw

```
            int num = 0;
            if (GM.CurrentPlayerBody.SetQuickbeltBasedOnStringName(quickbeltLayoutName, out num))
            {
                GM.Options.QuickbeltOptions.QuickbeltPreset = num;
                GM.Options.SaveToFile();
            }
        }
    }
    Dictionary<VaultElement, FVRPhysicalObject> dicElementToObj = new Dictionary<VaultElement, FVRPhysicalObject>();
    // Changed to ordered dictionary, so that KeyValuePairs added maintain their order.
    OrderedDictionary dicObjectsContainedInIndex = new OrderedDictionary();
    for (int o = 0; o < f.Objects.Count; o++)
    {
        VaultObject obj = f.Objects[o];
        if (IM.OD.ContainsKey(obj.Elements[0].ObjectID))
        {
            if (IM.HasSpawnedID(IM.OD[obj.Elements[0].ObjectID].SpawnedFromId))
            {
```

```
                else if (decodeAsLoadout)
                {
                    rootObj.transform.position = SpawnRelativeTo.position + Vector3.up * 0.3f * (float)o2;
                    rootObj.transform.rotation = SpawnRelativeTo.rotation;
                }
                else if (!decodeAsLoadout)
                {
                    rootObj.transform.position = obj2.Elements[0].PosOffset;
                    rootObj.transform.rotation = Quaternion.LookRotation(obj2.Elements[0].OrientationForward, obj2.Elem
                }
                if (obj2.IsContainedIn > -1)
                {
                    Debug.Log(rootObj.gameObject.name + " is contained in index " + obj2.IsContainedIn);
                    rootObj.SetActive(false);
                    dicObjectsContainedInIndex.Add(rootObj, obj2.IsContainedIn);
                }
                spawnedObjs[obj2.Index] = rootObj.GetComponent<FVRPhysicalObject>();
            }
        }
    }
    yield return null;
```

Code | Blame | 403 lines (399 loc) · 19.2 KB | Code 55% faster with GitHub Copilot | Raw

```
355                 FVRPhysicalObject fvrphysicalObject = dicElementToObj[vaultElement3];
356                 FVRQuickBeltSlot fvrquickBeltSlot = fvrphysicalObject.Slots[f.Objects[l].QuickbeltSlotIndex];
357                 spawnedObjs[l].SetQuickBeltSlot(fvrquickBeltSlot);
358             }
359         }
360     }
361     // This is where the newly ordered dictionary will add contained objects maintaing the order of the vault file.
362     foreach (KeyValuePair<GameObject, int> keyValuePair in dicObjectsContainedInIndex)
363     {
364         FVRPhysicalObject fvrphysicalObject2 = spawnedObjs[keyValuePair.Value];
365         fvrphysicalObject2.ContainOtherObject(keyValuePair.Key.GetComponent<FVRPhysicalObject>());
366     }
367     if (del != null)
368     {
369         List<FVRPhysicalObject> list = new List<FVRPhysicalObject>();
370         for (int m = 0; m < spawnedObjs.Length; m++)
```

Bug 2: After spawning a SaveableGunCase from vault file, opening, then closing it, the undeleted positions from the previous items inside the case persist.

Why: *Somehow*, the case newly spawned from the VaultFile somehow contains “extra” positions and rotations. After OpenCase() is called, the extra positions aren’t deleted since there are more positions than there are objects in m_containedObjects. Any future objects saved to this case will then spawn “at random” in any of the previous positions each time the case is opened.

How:

1. Modified lines of **OpenCase**
    a. Line 401: clear all positions after object decontainment loop
    b. Line 402: clear all rotations after object decontainment loop

We expect these two lists to be empty at the end of the OpenCase() call anyways. Adding this cleanup ensures that whatever messup happened during Vault save/spawn is corrected. Tests show that gun case operates as usual after opening the case with this patch.

```
387 private void OpenCase()
388 {
389   Debug.Log("Opening Case");
390   this.m_isCaseClosed = false;
391   for (int i = this.m_containedObjects.Count - 1; i >= 0; i--)
392   {
393     Vector3 vector = base.transform.TransformPoint(this.m_containedPositions[i]);
394     Quaternion quaternion = base.transform.rotation * this.m_containedRotations[i];
395     this.m_containedObjects[i].transform.SetPositionAndRotation(vector, quaternion);
396     this.m_containedObjects[i].gameObject.SetActive(true);
397     this.DecontainOtherObject(this.m_containedObjects[i]);
398     this.m_containedPositions.RemoveAt(i);
399     this.m_containedRotations.RemoveAt(i);
400   }
401   this.m_containedPositions.Clear();
402   this.m_containedRotations.Clear();
403 }
```