Character’s Equipped Items / Locked and Unlocked Items (Concept) Doc

Most of this implementation with respect to Team 2, can be found using the following link to the **ItemMap** script.

<https://github.com/ksu-cs-robotics/fall-2019-gdp-main-project-arcadedream/blob/team2-2/src/ArcadeDream/Assets/Team2/Scripts/ItemMap.cs>

Implementations with Respect to Each Team

**Team 3:** For your guys’ part, you simply want to drag and drop your prefabbed wearable gameObjects into the **ItemMap** script’s reference fields in the Editor. Here, you’ll see each item reference corresponds to a particular ID.

For example, let’s say you drag and drop some prefabbed piece of clothing into ‘ID0’ in the **ItemMap**. Let’s also say when we get a certain Player’s **equipmentHash** from the database during initialization, it says, among other things, that this Player is supposed to be wearing the item with ID = 0. In this case, whatever item that you dragged into ID0 is the actual item that will be initialized on that Player in-game. What items have what IDs in **ItemMap** is completely up to you. However, you’ll also see that there are 4 sections in **ItemMap**. What’s that all about?

So how that works, is each section (range of IDs) will correspond to a particular type of item. For example, all items referenced under Description 1 (ID0 – ID7), could be **hats**, and all items referenced under Description 2 (ID8 – ID15) could be **shirts**, and so on. This is just so we know what ID ranges correspond to what parts of the body. It’ll make sure that when we’re initializing the character, we’ll know not to put shirts on heads, hats on legs, etc.

**Team 4:** There will be two columns in the database for this: **equipmentHash**, and **permissionsHash**. Both will be 32-bit long unsigned integers.

How this All Works Together

Let’s start with the **equipmentHash**, and **permissionsHash** values in the database. What do these two values represent, and how?

First, let’s start with the ‘how’. Each bit in both variables correspond to an item with a certain ID. Both **equipmentHash**, and **permissionsHash** are 32 bits in length, and therefore, they can both represent a set of 32 items. The bit to item mapping between both variables is the same (the first bit of both map to ID0 in **ItemMap**). They differ only in what each of their bit’s values actually represent.

For **permissionsHash**, each bit represents the boolean value of whether or not the Player may equip an item with a certain ID. For example, if the first bit in **permissionsHash** was 1, that means the Player has unlocked the item with ID = 0, and therefore, can equip it. If the second bit were 0, this means the Player has not unlocked the item with ID = 1, and therefore, may not equip it. In the following example, we’ll look into this, and pretend that **permissionHash** is only 4-bits for simplicity.

Say our **permissionHash** is 5, or 0101 in binary. Our first bit in our 4-bit sequence is therefore ‘1’. But what does this actually mean?

Well, the first place in a binary sequence represents 2^0, but in our case, it’s actually 2^(Item ID). Since this ‘1’ resides in the 2^0 place, that means it corresponds to the item with ID = 0, or ID0 in **ItemMap**. The ‘1’ itself represents ‘true’, as in the Player has unlocked the item with ID = 0 or ID0 in **ItemMap,** and can therefore equip it. Alright so they player can wear ID0, but what about ID1?

Well the next bit in our 4-bit sequence is ‘0’, and this ‘0’ resides in the 2^1 place. Remember, the permission of a particular item is stored in the 2^(Item ID) place, and in this case (2^1), the item this bit represents is ID1 in **ItemMap**. Since this bit is ‘0’, that means the player has not unlocked the item with ID = 1 or ID1 in ItemMap, and therefore, may not equip it.

Using this principle, we can store the permissions of up to 32 items in our 32-it version of **permissionHash.** The 2^0 place corresponding with ID0 in ItemMap, the 2^1 place corresponding to ID1 in the ItemMap, and so on, and the actual bit value of those places representing if a player has unlocked the item or not. A ‘1’ means they have unlocked it, and may wear it, 0 means they have not, and may not wear it.

For **equipmentHash,** the idea is the same, however instead of each bit corresponding to whether or not can equip certain item, it’s whether or not a player has that item equipped. To use a similar example, if **equipmentHash** is 1001, this means the player is currently wearing items with the IDs 0, and 3 (ID0 and ID3 in **ItemMap**), and not wearing the items with the IDs 1, and 2 (ID1 and ID2 in **ItemMap**).

To summarize, **permissionHash** stores what items a player CAN wear, and **equipmentHash** stores what items a player IS wearing.

How to perform this programmatically? Take the GetPermission() function in ItemMap for example:

public static bool GetPermission(uint permissionsHash, uint itemID)

{

// Logical AND the permissionsHash with 2^ID, and convert that result to a bool

return Convert.ToBoolean(permissionsHash & ((uint)Math.Pow(2, itemID)));

}

Let break down how this works. To find out whether or not a player can equip an item, we must first get the **permissionsHash** of the player, as well as the item ID corresponding to the item they wish to wear.

First, we need to convert the ‘itemID’ to its binary representation with respect to **permissionsHash.** If itemID = 0, then 0001, if itemID = 1, then 0010, if itemID = 2, then 0100, etc. We do this by taking itemID, and raising 2 to the power of it’s value. We then Logical AND it with **permissionsHash.** What this essentially does is allow us to compare only a single bit between 2^itemID and **permissionsHash.**

For example, if **permissionsHash** is 0110, and itemID = 3, 2^3 = 1000, 0110 & 1000 = 0000. If this operation returns zero as it did in the latter example, we will convert that 0 to boolean, get ‘false’, and establish the player CANNOT equip / has not unlocked the item with ID = 3.

If, however, itemID = 2, 2^2 = 0100, 0110 & 0100 = 0100, or 4. When this 4 is converted to bool, it will evaluate to ‘true’, and establish the player CAN equip / has unlocked the item with ID = 2.

In another situation, if a player buys an item from the shop (unlocks it), it’s permission value needs to be updated in **permissionsHash.** We do this by taking the itemID, 2 for example, 2^2 = 0100, and a **permissionsHash** of 0001, Logical XOR them, and store the resulting value in the database as the updated **permissionsHash.** 0100 ^ 0001 = 0101 (new **permissionsHash**).

In the same example, if this player’s **equipmentHash** is 1001, i.e., they have 2 items, ID0 and ID3 currently equipped, and they want to put on the item they just unlocked / bought, we will take the itemID = 2, 2^2 = 0100, and Logical XOR it with **equipmentHash** and store the resulting value in the database as the updated **equipmentHash**. 0100 ^ 1001 = 1101 (new **equipmentHash**).

The values of **permissionsHash** will be used to initialize what items are available in the character creator, as well as what items can be bought from the shop.

The values of **equipmentHash** will be used to initialize the character with all of the items that they are already wearing.

This document is incomplete, and will be updated as need through the process of our implementation.