2020년 2학기 한림대학교 콘텐츠IT

AR을 이용한 Android 라이브 배경화면

나만의 작은방 (My Little Wallpaper)

개발 정보

개발 기간

2020/ 09/ 16 ~ 2020/ 12/ 10

사용 툴, 언어

Unity, ARCore, C#

타겟 플랫폼

Android

개발 인원

- 기획, 클라이언트 개발 20145327 오세영 (팀장)
- 디자인, 클라이언트 개발 20145322 서정훈

지도교수

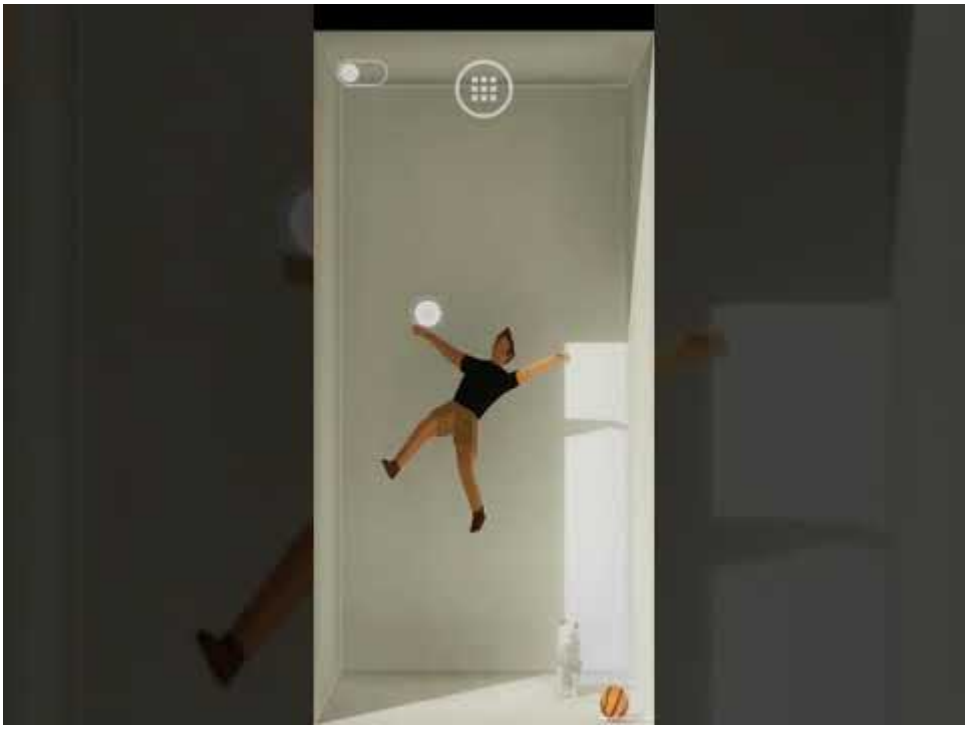
- 이정 (한림대 소프트웨어융합대학)

앱 정보

개요

- 기존에 간단한 이미지나 동영상으로만 꾸밀 수 있던 스마트폰의 배경화면을 3D소품들을 이용해 꾸밀 수 있다.

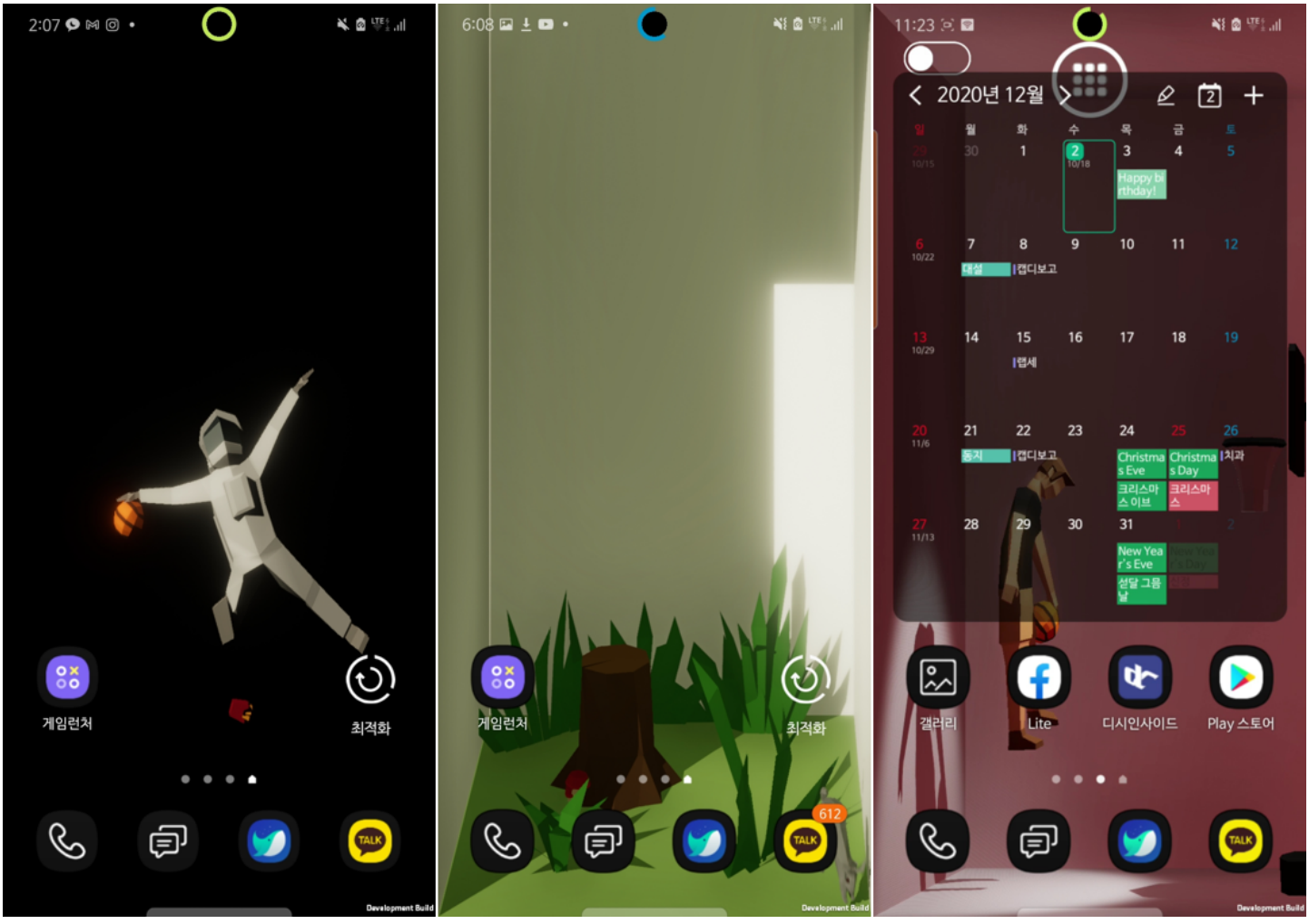
시연 영상



기능 설명

- 배경화면 설정

앱자체를 배경화면으로 설정할 수 있다.

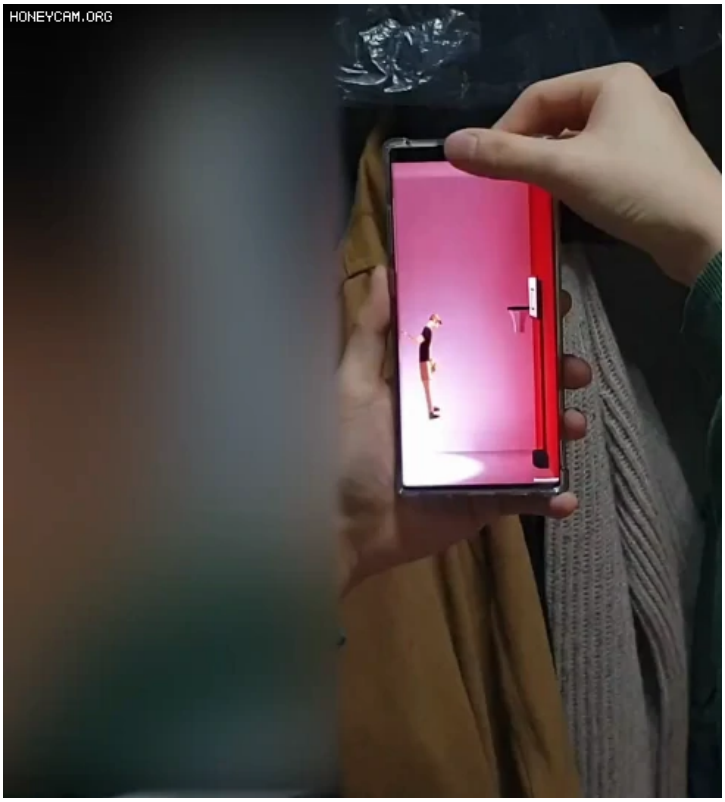


- 편집모드

해당 앱은 수십가지의 다양한 로우폴리 3D모델들을 제공한다. 그리고 위치, 각도, 크기, 색상 등 소품의 상세한 속성들을 사용자가 원하는대로 설정이 가능하다.

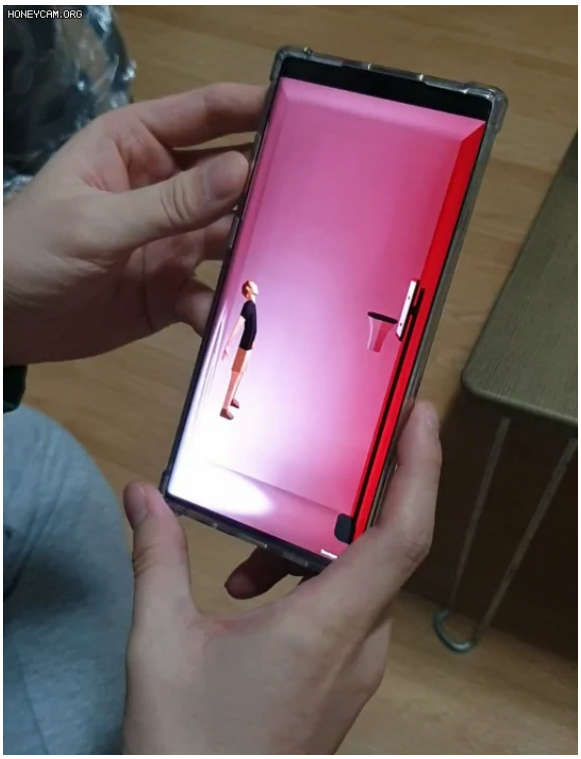
- 안면인식

전면부 카메라를 이용해, 사용자의 안면을 인식한다. 사용자의 안면 위치에 따라 방이 회전하며, 마치 실제 공간을 바라보는 듯한 착시를 유도할 수 있다.



- 자이로스코프

휴대폰에 내장된 자이로스코프센서를 이용해, 방을 회전한다. 마찬가지로, 실제 공간을 바라보는 듯한 착시를 유도할 수 있다.



계층 구조(Hierarchy) 코드 설명

해당 앱에는 오브젝트(소품)을 배치하기 위한 계층 구조(Hierarchy) UI가 존재한다.

실제 앱에서 유저가 보게되는 계층 구조 *UI*

전체적인 계층 구조 초기화 코드

앱이 시작되고 계층 구조 정보를 초기화하는 구간이다.

```
        Debug.Log("Hierarchy 업데이트 시작: " + System.DateTime.Now.ToString("hh:mm:ss"));

        //-현재 상황-
        //저장소에서 불러온 아이템 정보들을 기반으로 아이템 풀에서 오브젝트를 불러와 각자의 위치에 배치시킴
        //또한, 아이템의 주소정보도 불러와 '슬롯'을 제외한 모든 아이템들에게 각자의 계층구조 주소(경로)를 적용함

        //'슬롯'이란?
        //소품내부에 기본적으로 제공되는 파츠
        //ex) 사람: 머리, 허리, 오른손, 왼손, 오른팔꿈치, 왼팔꿈치, 왼발, 오른발, 왼쪽무릎, 오른쪽무릎
        //이들 또한 자신들의 주소를 가지고 있음

        UpdateHierarchyItemList(); // 먼저 앱 내에 배치된 모든 아이템과 슬롯들의 노드UI오브젝트를 계층구조UI에 생성하고, 배치되어있는 아이템 계층구조를 참고하여 업데이트 함

        GameManager.instance.roomInfo.placedItems.Sort((x1, x2) => x1.AdressDepth.CompareTo(x2.AdressDepth)); //주소 깊이가 얕은 순서로 아이템 리스트 정렬
        //깊이가 얕은 순서로 정렬하는 이유는 슬롯 때문임
        //슬롯은 데이터를 저장할 때, 정보가 저장되지 않기 때문에
        //데이터를 불러올 때, 최초 주소값을 자신의 이름으로 초기화 함
        //때문에, 슬롯이 없는(깊이가 얕은)순서로 계층구조를 업데이트해야함

        for (int i = 0; i < GameManager.instance.roomInfo.placedItems.Count; i++)
        {
            PlaceableItem item;
            if ((item = GameManager.instance.roomInfo.placedItems[i] as PlaceableItem) && item.itemAdress.Contains("/"))
            {
                ItemBase temp;
                Transform parentItem = (temp =
                    GameManager.instance.roomInfo.placedItems.Find((compareItem) => compareItem.IsParent(item))) ? temp.transform : GameManager.instance.roomContents.transform;
                // 주소를 비교하여 부모아이템 검색
                // 부모가 없다면 방을 부모로 삼음

                item.transform.SetParent(parentItem); // 탐색하여 반환된 아이템을 부모로 설정

                item.ResetAdress();
                GameManager.SetAdressItemToChild(item, ref item.itemAdress); // 해당 아이템의 하위 아이템들의 주소를 재설정
                //주소를 재설정 하는 이유: 슬롯의 주소를 업데이트 해주어야 하기 때문
            }
        }
        yield return new WaitForEndOfFrame();//순서3

        // 모든 슬롯들의 주소도 초기화 되었을 뿐만아니라, 실제 배치된 아이템들의 계층구조 또한 자리를 잡았음
        // 이제 계층구조UI를 실제 아이템들의 계층구조에 맞게 업데이트만 해주면 됨

        UpdateHierarchyItemList(); // 계층구조UI의 노드UI들을, 배치되어있는 아이템 계층구조를 참고하여 업데이트 함
        Debug.Log("Hierarchy 업데이트 완료: " + System.DateTime.Now.ToString("hh:mm:ss"));
```

다음 아래의 메소드들은 위 구간에서 사용된 메소드들이다.

UpdateHierarchyItemList

해당 메소드는 실제 배치된 아이템들의 계층 구조를 계층 구조 UI에 동기화 시킬 때 사용한다.

```
    public void UpdateHierarchyItemList()
    {
        currentHierarchyIndex = 4; // 계층구조UI 필터 (0= 식물, 1=동물, 2=사물, 3=조명, 4=전체)

        InitHierarchyItemList(); // 계층구조UI의 노드UI 전부 삭제하고, 배치된 아이템 전부를 계층구조UI에 노드UI로 등록함

        HierarchyArrangement(); // 노드UI 정렬
    }
    참조 1개
    public void InitHierarchyItemList()
    {
        DestroyChilds(hierarchyList); // 계층구조UI의 노드UI 전부 삭제

        var sourceList = GameManager.instance.roomInfo.placedItems;
        for (int i = 0; i < sourceList.Count; i++)
        {
            var item = sourceList[i];
            item.itemOnHierarchy = Instantiate(hierarchyContentPrefab, hierarchyList).GetComponent<HierarchyItemContent>();
            item.itemOnHierarchy.item = item;
        } // 배치된 아이템 전부를 계층구조UI에 노드UI로 등록

    }
    참조 1개
    public void HierarchyArrangement()
    {
        for (int i = 0; i < hierarchyList.childCount; i++)
        {
            var hierarchyItem = hierarchyList.GetChild(i).GetComponent<HierarchyItemContent>();
            //Debug.Log("정렬중: "+hierarchyItem.item.name+"₩n주소: "+ hierarchyItem.item.itemAdress);
            ItemBase parentItem;
            if (parentItem = GameManager.GetParentItem(hierarchyItem.item.transform)) // 배치된 아이템의 부모 아이템을 탐색하고, 부모가 있다면
            {
                hierarchyItem.transform.SetParent(parentItem.itemOnHierarchy.childSlot); //  계층구조UI에서 해당 아이템의 노드UI를, 부모 아이템의 노드UI 자식으로 배치
                hierarchyItem.parentItem = parentItem.itemOnHierarchy;
                parentItem.itemOnHierarchy.HideChilds();
                i--;
            }
        }
        UpdateHierarchySize();
    }
```

GetParentItem

```
888     public static ItemBase GetParentItem(Transform item)
889     {
890         //재귀 함수를 통한 계층구조 탐색
891         ItemBase result;
892         if (result = item.parent.GetComponent<ItemBase>())// 아이템이거나 슬롯일 경우 반환
893             return result;
894         else if (!item.parent.GetComponent<RoomManager>())// ItemBase 컴포넌트가 존재하지 않는 일반 게임 오브젝트일 경우 계속 탐색
895             return GetParentItem(item.parent);
896         else                                            // 부모가 없을 경우 null 반환
897             return null;
898     }
```

IsParent, ResetAdress

```
        참조 4개
125     public void ResetAdress()
126     {
127         if (this is PlaceableItem) //아이템일 경우 주소를 ID로 초기화
128         {
129             itemAdress = ID.ToString();
130         }
131         else                        //슬롯일 경우 주소를 슬롯명으로 초기화
132             itemAdress = itemName;
133     }
        참조 1개
134     public bool IsParent(ItemBase childItem)
135     {
136         //Debug.Log("나는 누구?: " + itemName + ", " + ID + "₩n내 주소: " + itemAdress + "₩비교 주소: " + childItem.itemAdress);
137         if (this == childItem) return false;
138         string childItemAdressTemp = childItem.itemAdress;
139
140         //Debug.Log("나는 부모인가?: " + itemName + ", " + ID + "₩n내 주소: " + itemAdress + "₩n비교 주소: " + childItemAdressTemp);
141
142         string compareAdress = childItemAdressTemp.Replace("/" + childItem.ID.ToString(), "");
143         if (compareAdress == itemAdress)
144         {
145             //Debug.Log("나는 부모다!" + itemName + ", " + ID);
146             return true;
147         }
148         return false;
149     }
```

SetAdressItemToChild

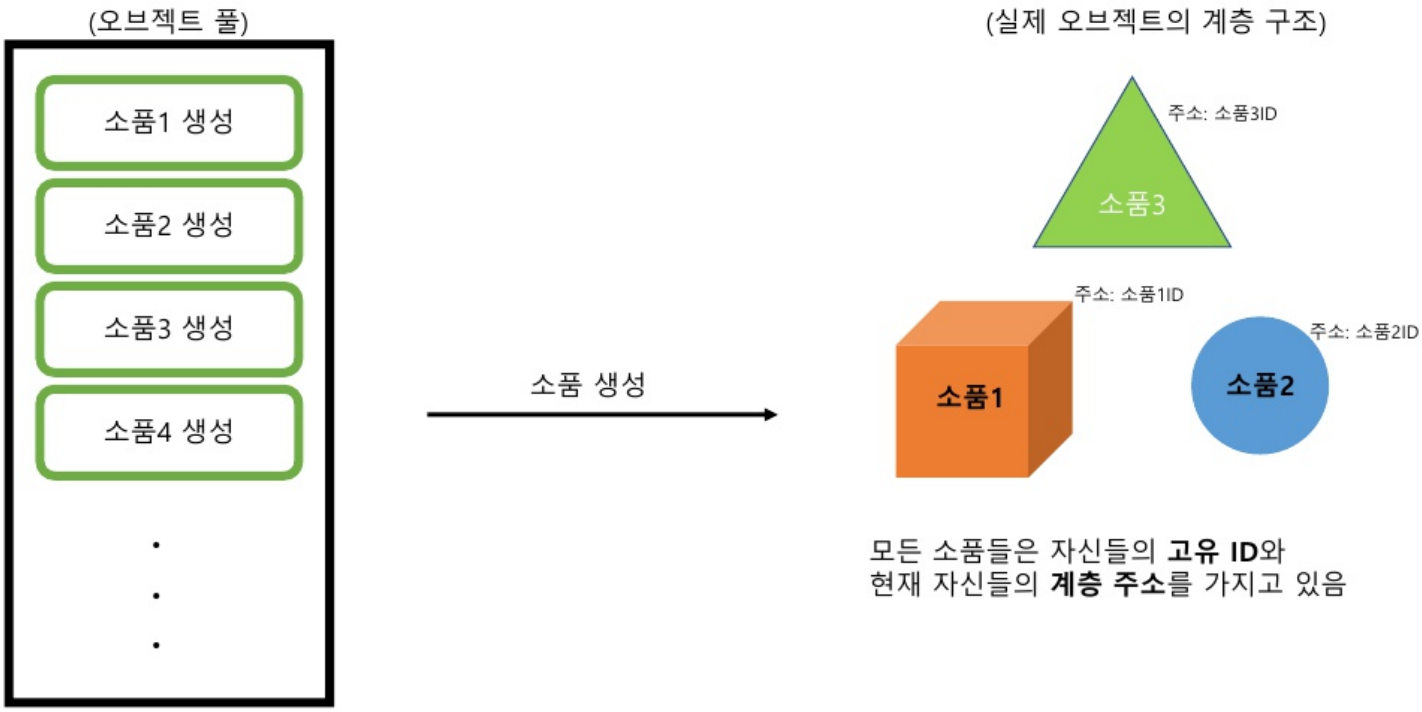
DFS탐색을 이용하여, 모든 아이탬들의 주소를 업데이트시켜준다.

```
930     public static void SetAdressItemToChild(ItemBase item, ref string adress)
931     {
932         //재귀 함수를 통해 계층구조의 하위방향으로 탐색 및 주소 설정
933         if (item.itemOnHierarchy.childSlot.childCount > 0) // 유니티의 Hierarchy는 다중 트리 구조이다. 때문에 자식이 여러개가 존재할 수 있다. 해당 사항을 고려하여 모든 자식에게 재귀 함수를 실행
934         {
935             for (int i = 0; i < item.itemOnHierarchy.childSlot.childCount; i++)
936             {
937                 var childItem = item.itemOnHierarchy.childSlot.GetChild(i).GetComponent<HierarchyItemContent>().item;
938                 childItem.ResetAdress();
939                 SetAdressItemToChild(childItem, ref childItem.itemAdress);
940             }
941         }
942         Transform parent = item.transform.parent;
943         ItemBase parentItem;
944         if (parentItem = item.transform.parent.GetComponent<PlaceableItem>())   // 부모가 아이템일 경우 부모의 ID를 주소에 추가, 이후 계속 탐색
945         {
946             adress = adress.Insert(0, parentItem.ID.ToString() + "/");
947             SetAdressItemToParent(parent, ref adress);
948         }
949         else if (parentItem = item.transform.parent.GetComponent<ItemBase>())   // 부모가 슬롯일 경우 부모의 슬롯명을 주소에 추가, 이후 계속 탐색
950         {
951             adress = adress.Insert(0, parentItem.itemName + "/");
952             SetAdressItemToParent(parent, ref adress);
953         }
954         else if (!item.transform.parent.GetComponent<RoomManager>())           // ItemBase 컴포넌트가 존재하지 않는 일반 게임 오브젝트일 경우 계속 탐색
955         {
956             SetAdressItemToParent(parent, ref adress);
957         }
958         else                                                                    // 루트 아이템 일경우 탐색 종료
959             return;
960     }
```

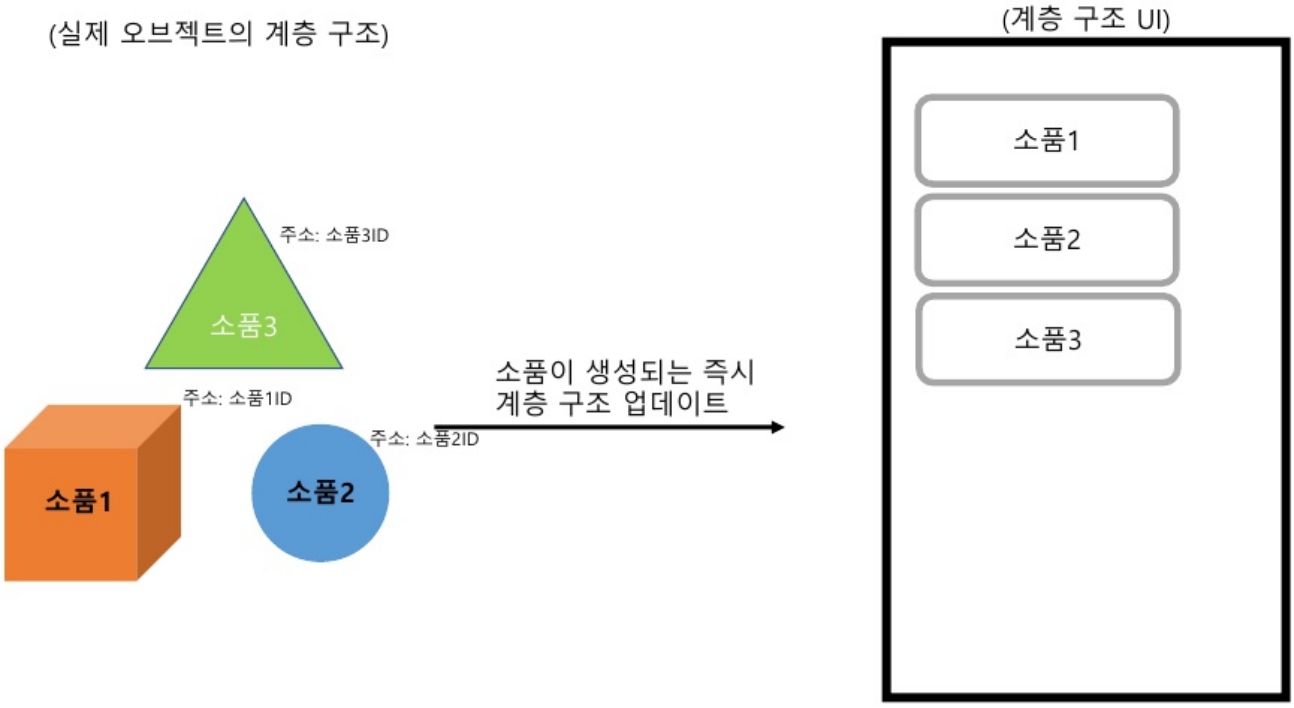
계층 구조(Hierarchy) 원리

해당 계층 구조의 원리는 다음과 같다.
(원활한 설명을 위해 '슬롯'개념은 제외)

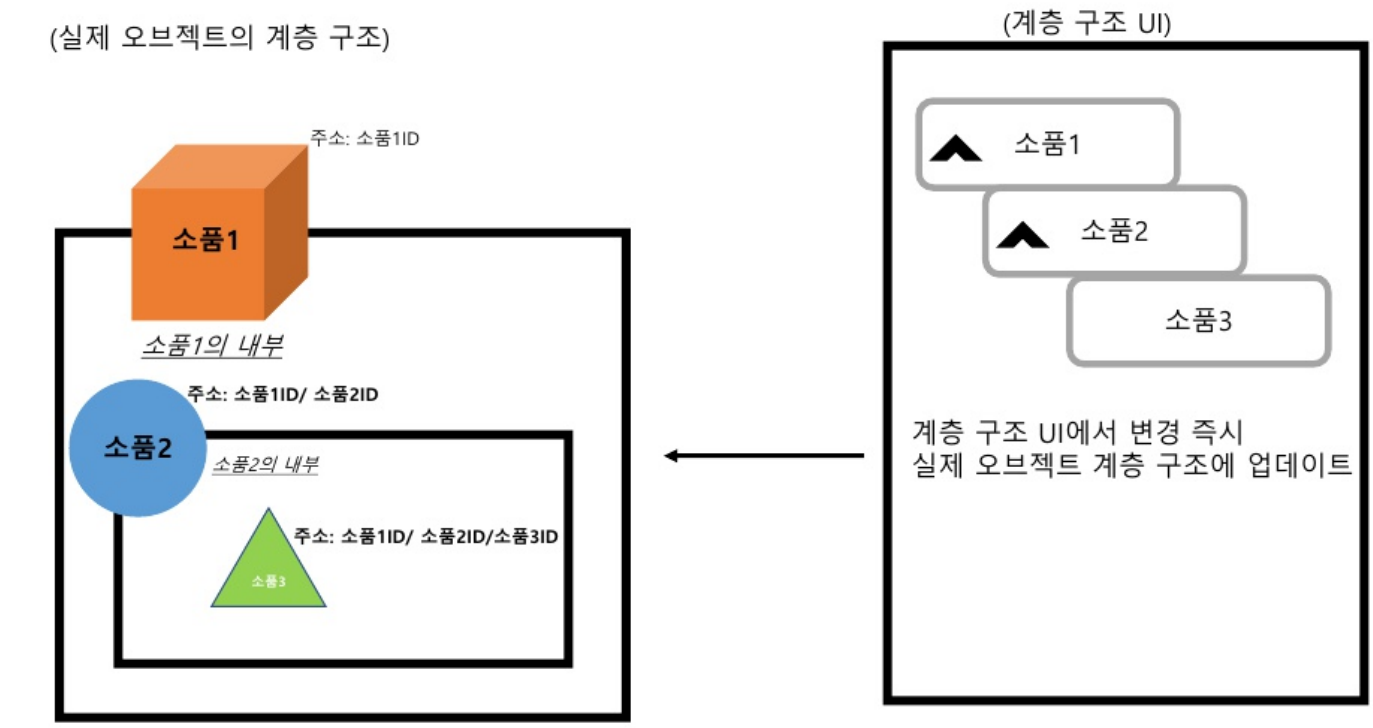
먼저 원하는 오브젝트를 선택하여 아래의 그림 처럼 방내부에 생성, 배치하면,



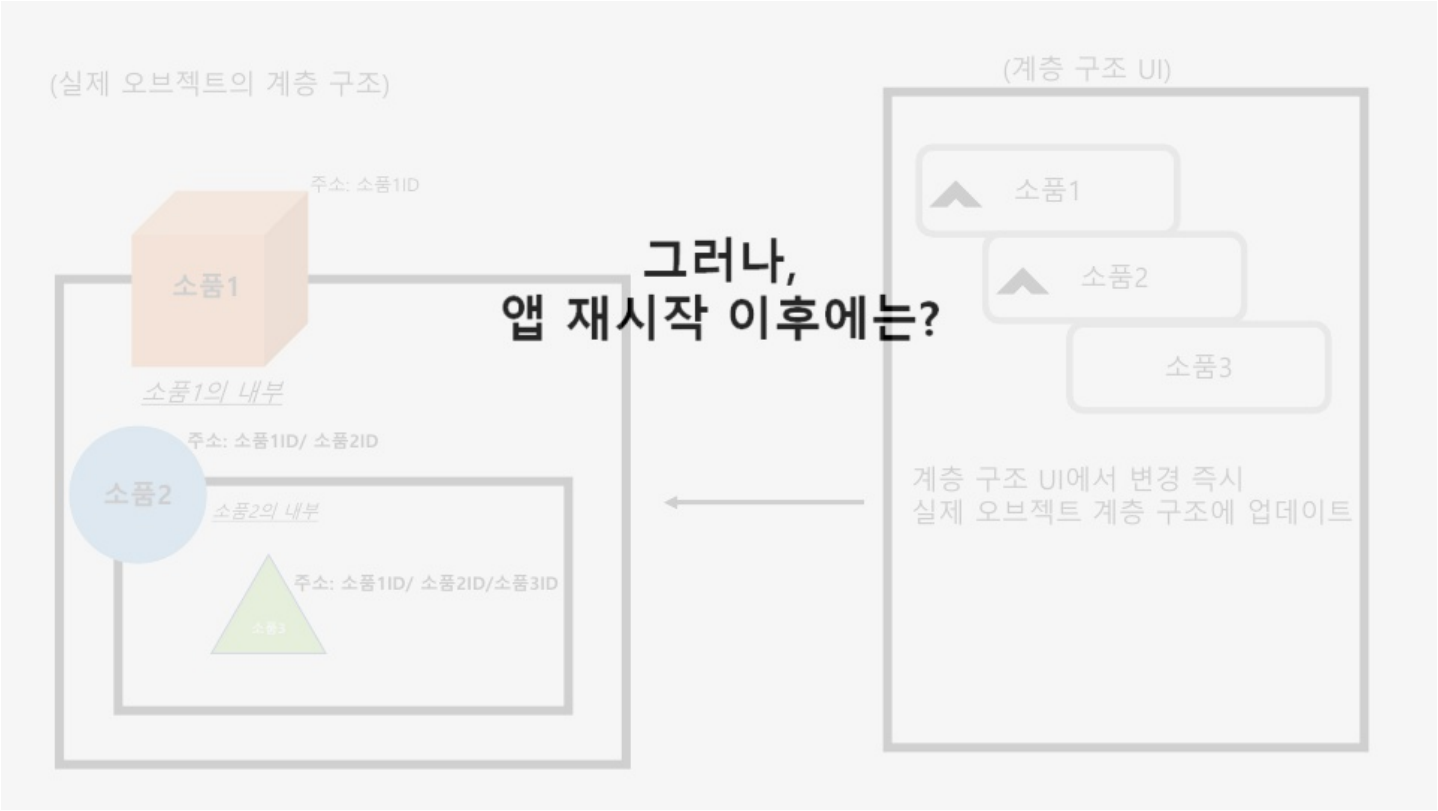
생성과 즉시 계층 구조 UI에 해당 정보를 가진 라벨이 생성된다.



또한 UI 에서 계층 구조 변경 시, 곧 바로 실제 오브젝트의 계층 구조도 같이 업데이트 된다.



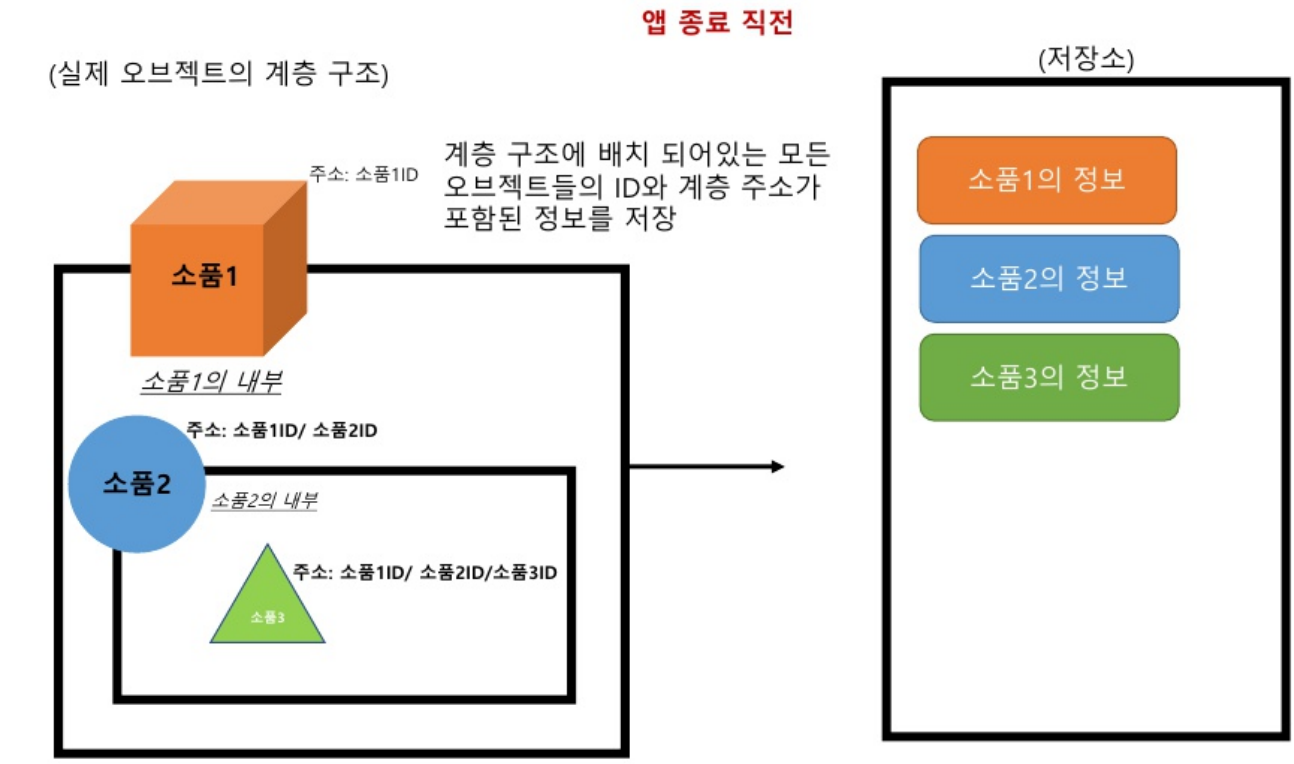
이렇게 완성된 방의 정보는 사용자 기기에 저장되고,
앱이 재실행 되어도 방의 정보를 그대로 불러 오게 된다.



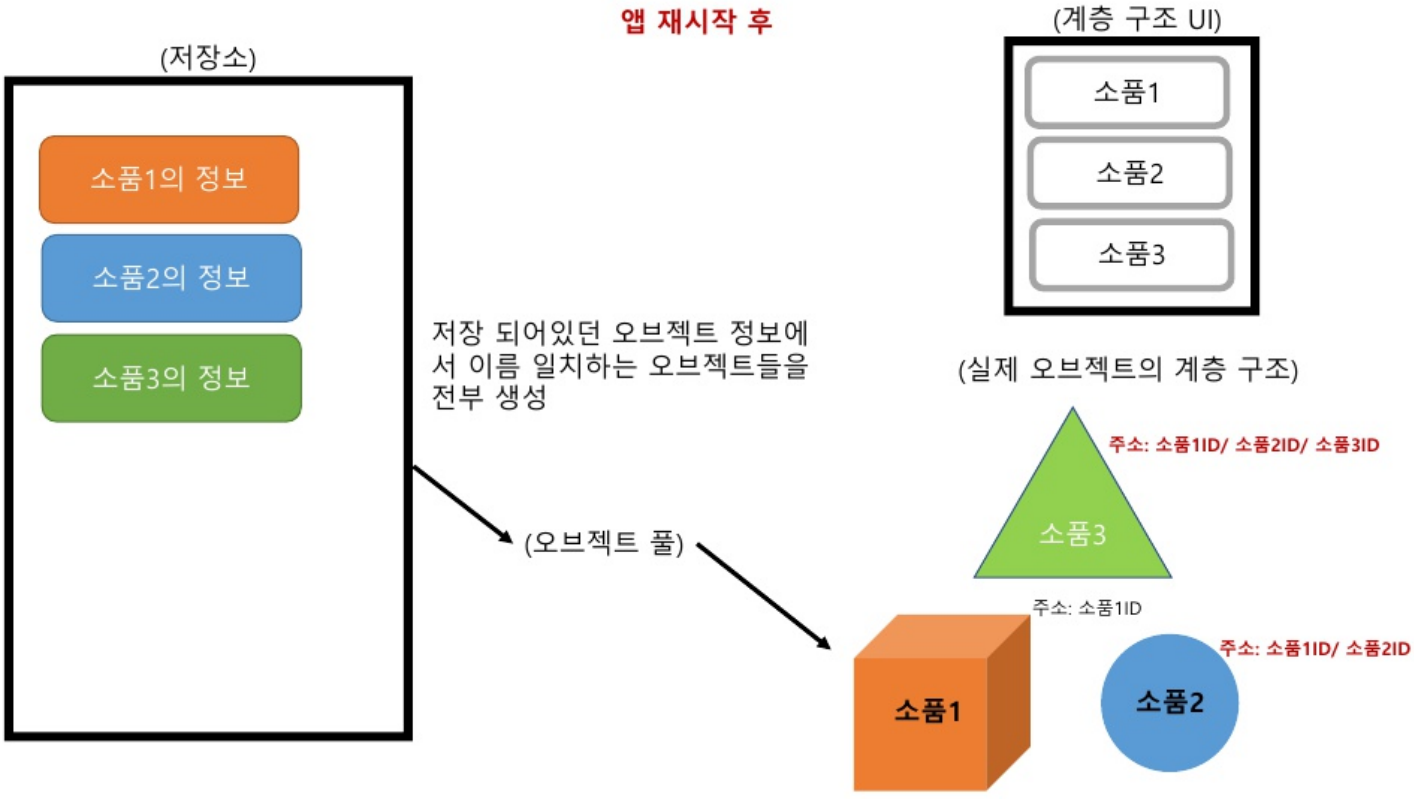
그렇다면, 앱 재시작 이후의 과정은 어떻게 될까?

해당 과정을 설명하기 위해서는, 앱 종료 직전부터 설명해야한다.
먼저 앱이 종료되면, 방에 배치된 모든 오브젝트들의 정보(ID, 계층 주소, 오브젝트 종류 등...)를 저장한

다.



이후 앱을 재시작하게 되면, 저장되어 있던 정보들을 불러와 해당 정보에 맞는 오브젝트들을 생성, 배치한다.



현재 배치된 오브젝트들은 ID, 이름, 종류, 주소값 등은 정상적으로 불러와 졌지만,
정보만 가지고 있을뿐 실제로 그 어떠한 상호 계층 구조 관계를 가지고 있지 않은 상태다.

계층 구조 업데이트에는 오브젝트의 주소를 이용한다.
오브젝트들은 서로의 주소에서 자신의 ID를 제외한 string값을 비교하여, 계층 구조 관계를 업데이트한

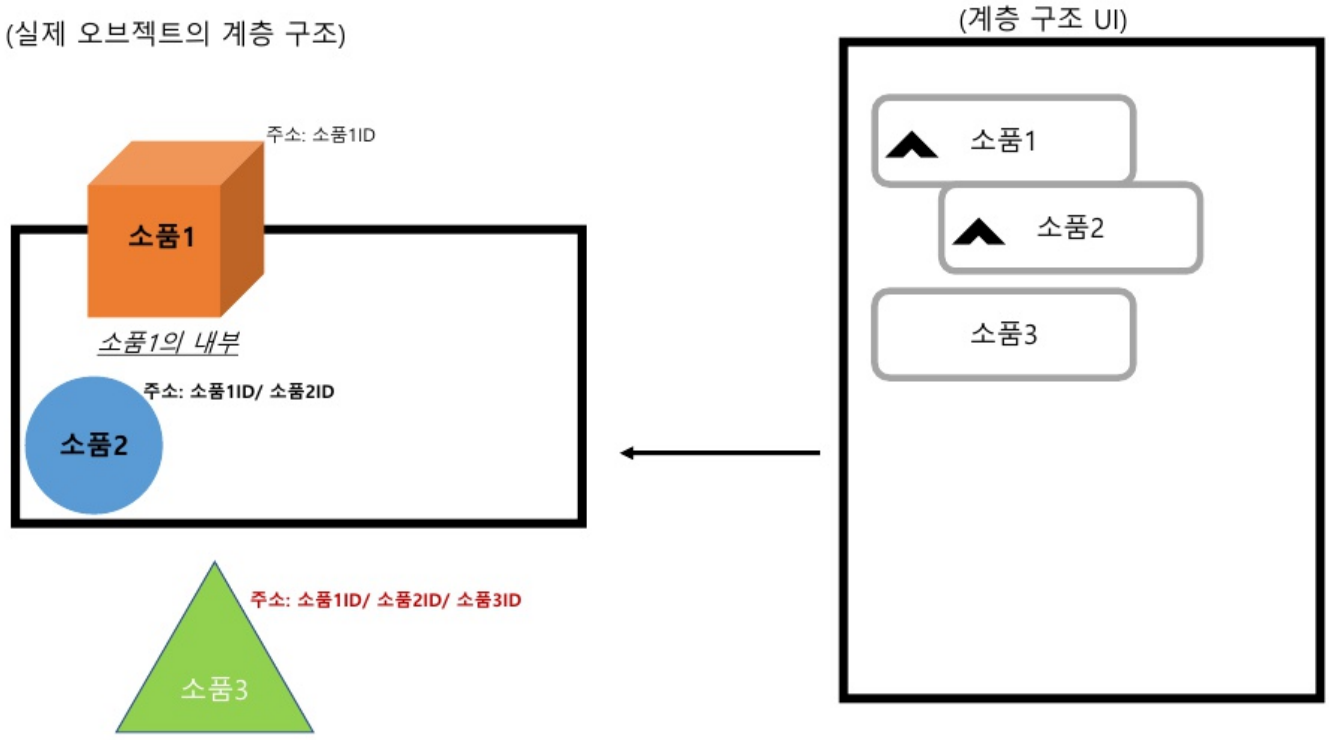
다.

소품2의 주소: 소품1ID/ 소품2ID

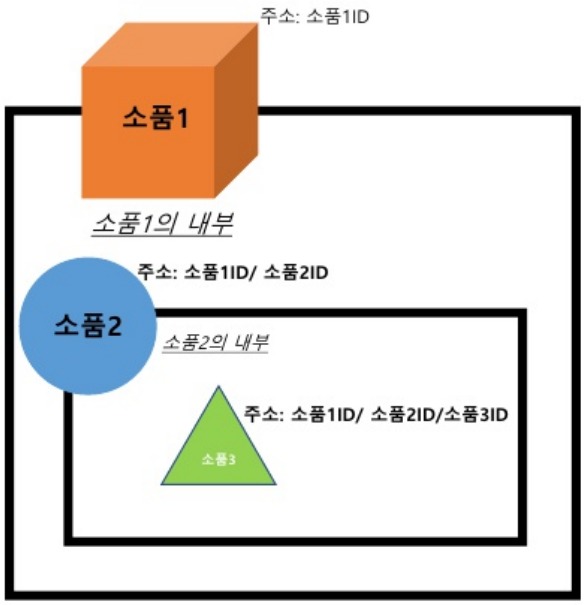
==

소품3의 주소: 소품1ID/ 소품2ID/~~소품3ID~~

∴ **소품2**는 **소품3**의 부모 이다.



(실제 오브젝트의 계층 구조)



(계층 구조 UI)