```
    private void ParentObjectsWithinColliders()
    {
        if (GUILayout.Button("Create Section Prefabs"))
        {
            // 0) reset lists in case of scene change
            _sectionCreatorsToCheckOverlap.Clear();
            _sectionCreatorsStructured.Clear();
            // 0.1) adjust scene string so level section don't have "Work" in their names
            AdjustSceneString();

            // 1) Find the blockout Parent
            BlockoutParent blockoutParent = FindObjectOfType<BlockoutParent>();
            if (blockoutParent == null)
            {
                Debug.LogWarning("Could not find a BlockoutParent script in the scene!");
                return;
            }

            // 2) Find the sectionCreatorPrefabs and unpack them
            _sectionCreatorsInScene = FindObjectsOfType<LevelSectionCreator>().ToList();
            for (int i = 0; i < _sectionCreatorsInScene.Count; i++)
            {
                if (PrefabUtility.IsAnyPrefabInstanceRoot(_sectionCreatorsInScene[i].gameObject) == true)
                {
                    // this being registered as a user action allows it to be Undo-d
                    PrefabUtility.UnpackPrefabInstance(_sectionCreatorsInScene[i].gameObject, PrefabUnpackMode.OutermostRoot,
                        InteractionMode.UserAction);
                }
            }

            // 2.1) assign starting section
            LevelSectionCreator sectionZero = null;
            int amountOfStartingSectionsFound = 0;
            for (int i = 0; i < _sectionCreatorsInScene.Count; i++)
            {
                if (_sectionCreatorsInScene[i].IsStartingSection)
                {
                    sectionZero = _sectionCreatorsInScene[i];
                    amountOfStartingSectionsFound += 1;
                }
            }
            if (amountOfStartingSectionsFound == 0)
            {
                Debug.LogWarning("No Starting Section found! make sure atleast 1 has the bool IsStart 'checked'");
                // undo-ing the unpacking of the section creators
                Undo.PerformUndo();
                return;
            }
            else if (amountOfStartingSectionsFound > 1)
            {
                Debug.LogWarning("Found more than 1 Starting Section! make sure only 1 has the bool IsStart 'checked'");
                // undo-ing the unpacking of the section creators
                Undo.PerformUndo();
                return;
            }

            // 2.2) use list of sections we need to check for collision
            for (int i = 0; i < _sectionCreatorsInScene.Count; i++)
            {
                _sectionCreatorsToCheckOverlap.Add(_sectionCreatorsInScene[i]);
            }
            // 2.3) Remove starting section from sectionCreatorsToCheck
            _sectionCreatorsToCheckOverlap.Remove(sectionZero);
            // 2.4) Add SectionZero to structured list
            _sectionCreatorsStructured.Add(sectionZero);
            // 2.5) use the starting section to progressively find following sections by checking the heads
            FindFollowingSectionCreator(sectionZero);

            // 3) Next, get all objects, duplicate them, add to temp list, check tempList objects for inside of bounds, parent objects
            // 3.1) duplicate the blockout_parent
            BlockoutParent blockoutCopy = Instantiate(blockoutParent);
            // 3.2) disable the original blockout
            blockoutParent.gameObject.SetActive(false);
            // 3.3) add all children of the copy to a temp list
            List<GameObject> tempList = new List<GameObject>();
            ParentAllTheThings(tempList, blockoutCopy.transform);

            // 4) iterate over each sectionCreator...
            for (int i = 0; i < _sectionCreatorsStructured.Count; i++)
            {
                // 4.1) iterate over each collider...
                foreach (BoxCollider coll in _sectionCreatorsStructured[i].CollidersDefiningMySection)
                {
                    // 4.2) parent any objects that fall within the bounds (only check first children of the blockout)
                    List<GameObject> objectsAdded = new List<GameObject>();
                    foreach (GameObject obj in tempList)
                    {
                        if (coll.bounds.Contains(obj.transform.position))
                        {
                            // check if the object has pickups its children
                            // if so -> add the children instead of the obj itself
                            List<PickUp> pickupsFound = obj.GetComponentsInChildren<PickUp>(true).ToList();
                            if (pickupsFound.Count > 0)
                            {
```

```
                                GameObject pickupChild = null;
                                for (int k = 0; k < pickupsFound.Count; k++)
                                {
                                    pickupChild = pickupsFound[k].gameObject;

                                    pickupChild.transform.SetParent(_sectionCreatorsStructured[i].Section.PickupsParent.gameObject.transform);
                                    objectsAdded.Add(pickupChild);
                                }
                                continue;
                            }

                            // any other gameobject that does not have pickups in its children
                            objectsAdded.Add(obj);

                            // differentiate between pickups and environment
                            if (obj.TryGetComponent(out PickUp pickUp))
                            {
                                obj.transform.SetParent(_sectionCreatorsStructured[i].Section.PickupsParent.gameObject.transform);
                            }
                            else
                            {
                                obj.transform.SetParent(_sectionCreatorsStructured[i].Section.ParentEnvironment.transform);
                            }
                        }
                    }

                    // 4.3) remove previously added objects from tempList (performance tweak)
                    foreach (GameObject obj in objectsAdded)
                    {
                        if (tempList.Contains(obj))
                        {
                            tempList.Remove(obj);
                        }
                    }
                    objectsAdded.Clear();
                }

                // 4.4) name our section something fitting
                _sectionCreatorsStructured[i].Section.name = "PV_LevelSection_" + _adjustedSceneString + i;
                _sectionCreatorsStructured[i].Section.ParentEnvironment.name = "Environment_" + i;
                _sectionCreatorsStructured[i].Section.PickupsParent.gameObject.name = "Pickups_" + i;

                // 4.5) log data regarding the section into the console
                LogCurrentSectionData(_sectionCreatorsStructured[i].Section);
            }
            LogAllData();

            Debug.Log("finished parenting blockout");

            // 5) get correct directory dependant on our scene
            string levelIndexString = DecodeSceneString()[0].ToString();
            string levelSceneIndexString = DecodeSceneString()[1].ToString();
            string localPath = _localPathPrefix + levelIndexString + _localPathMidfix + levelSceneIndexString + _localPathSuffix;
            // 5.1) delete folder and its contents, then Re-create it
            if (Directory.Exists(localPath))
            {
                Directory.Delete(localPath, true);
            }
            Directory.CreateDirectory(localPath);

            // 6) create prefab of each section
            for (int i = 0; i < _sectionCreatorsStructured.Count; i++)
            {
                Section sectionOfInterest = _sectionCreatorsStructured[i].Section;

                // 6.1) get the Section of interest, Unparent it (prefab will remember it's world position this way)
                GameObject objectToPrefabify = sectionOfInterest.gameObject;
                objectToPrefabify.transform.SetParent(null);
                // 6.2) Create the new Prefab.
                PrefabUtility.SaveAsPrefabAsset(objectToPrefabify, localPath + objectToPrefabify.name + ".prefab");
                // 6.3) Destroy all the children in environment and pickups (wipe clean for a scene reset) of current Section
                for (int j = sectionOfInterest.ParentEnvironment.transform.childCount - 1; j >= 0; j--)
                {
                    DestroyImmediate(sectionOfInterest.ParentEnvironment.transform.GetChild(j).gameObject);
                }
                for (int j = sectionOfInterest.PickupsParent.transform.childCount - 1; j >= 0; j--)
                {
                    DestroyImmediate(sectionOfInterest.PickupsParent.transform.GetChild(j).gameObject);
                }

                // 6.4) Re-parent it to its creator
                objectToPrefabify.transform.SetParent(_sectionCreatorsStructured[i].transform);
            }

            // 7) reset the scene by destroying duplicates
            // 7.1) Re-enable the old blockout
            blockoutParent.gameObject.SetActive(true);
            // 7.2) destroy copied blockout (should ideally be empty object after all of its children have been re-parented)
            DestroyImmediate(blockoutCopy.gameObject);

            // 8) undo-ing the unpacking of the section creators
            Undo.PerformUndo();
        }
    }

```