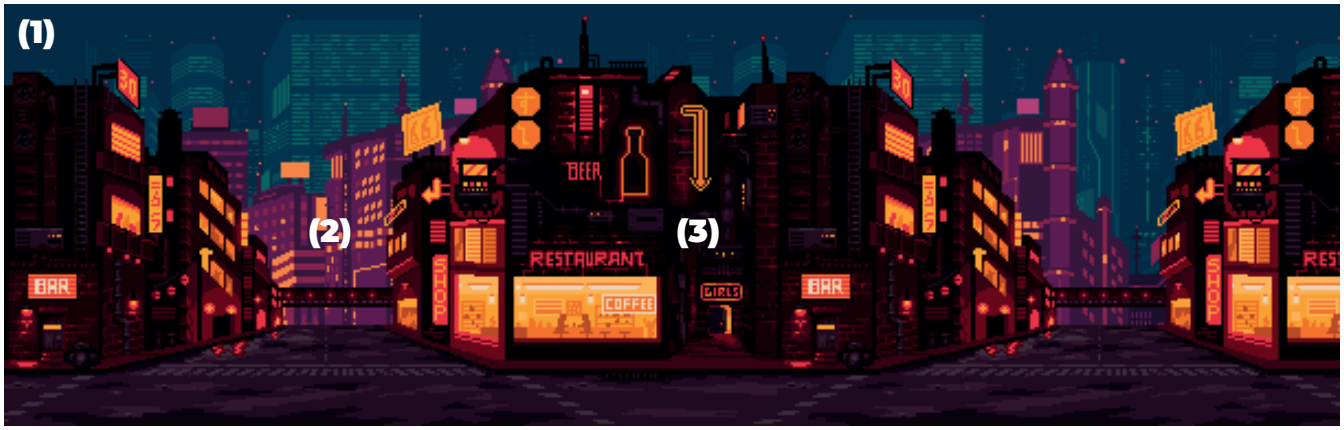
Comment créer un effet parallax avec un arrière plan 2D infini ?

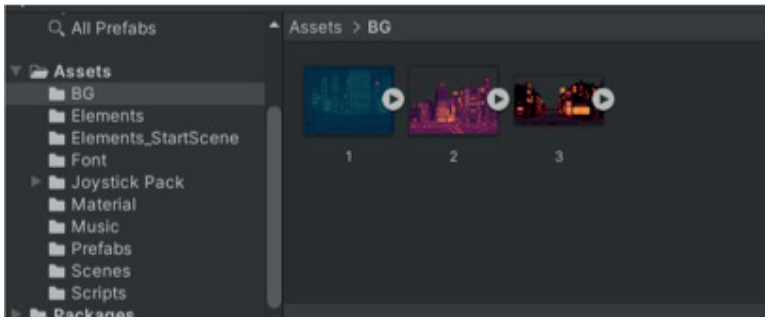
Pour créer un effet parallaxe avec un arrière-plan 2D, nous avons besoins de plusieurs « game-object ». Ces game-object sont des images d'arrière plan que vous allez importer dans votre scène.. Par exemple nous avons dans notre cas besoin du fond de la ville (1), des bâtiments_violet(2) puis des bâtiments-noir(3).



1 Importer vos images dans votre projet

Importer vos images dans la fenêtre «***Project***» dans Unity, en prenant soin de les nommer.

Nous importons dans notre cas nos 3 images, dans un dossier «Game object»



2 Imiter une perspective en ajoutant nos images à la scène

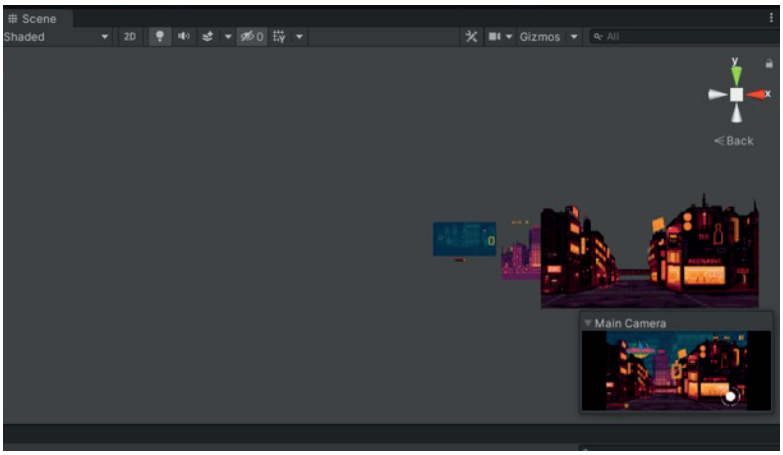
Nous allons imiter une perspective. Pour rappel plus un objet est éloigné sur l'axe Z, plus il se déplacera lentement dans l'arrière-plan. Il faut donc placer vos objets dans l'ordre souhaité dans la scène en prenant en compte cet élément.

- Dans notre cas nous ajoutons dans la fenêtre «***Hierarchy***» chaque image. En premier le fond de la ville (1), les bâtiments-violet(2), et les bâtiments-noir(3).

- Par la suite il faut configurer chaque objet de notre arrière-plan. Pour le game-objet contenant le ciel et le fond de la ville nous souhaitons qu'il se déplace doucement.

  - Appuyez donc dans la fenêtre « ***Hierarchy*** » sur votre objet le plus lointain dans votre scène (1) , et modifier dans la fenêtre « **Inspector** » la valeur de l'axe Y. Nous pouvons par exemple mettre **la valeur 80**. Il suffit ensuite de répéter l'action en cliquant sur l'objet suivant **dans la perspective**, et mettre une valeur inférieure à celle précédente.

  Pour l'exemple nous allons dans notre cas mettre 40 pour nos bâtiments-violet (2), puis 30, pour nos derniers bâtiments-noir(3). Si maintenant vous basculez sur **la vue 2D**, vous pourrez voir que vos objets sont décalés les uns par rapport aux autres (voir image). .



ATTENTION, plus un objet est éloigné de la caméra plus il se déplacera à la vitesse de la caméra, plus il est proche de la caméra, moins il bougera.

3 Créer un script #C

Nous pouvons désormais créer un script #C, dans la fenêtre « projet » et le dossier « script ». Nommez le «backgroundLoop ».

Quel rôle va jouer le script ?

Le script que nous allons ensuite créer va cloner nos objets(ville, et bâtiments) les uns à la suite des autres en prenant en compte une dimension temporel. Cette dimension temporelle va ainsi permettre de créer un léger décalage de mouvement entre les images et de créer l'effet parallaxe souhaité. De plus le script prend en compte la position de la caméra pour générer automatiquement le déplacement des objets.

Lorsqu'un élément est cloné il se place en enfant de l'objet initial. L'objet de base est alors « dépouillé » de son Sprite Renderer et son unique travail est alors de garder tous les clones ensemble de sorte que la caméra puisse suivre la scène.

Le script permet d'établir des « points » de vérification détectant si la caméra est sur le point de dépasser le bord du premier ou dernier objet « enfant » de la scène (Background 1, ou Background 3). De ce fait, si la caméra bouge à droite de l'écran, le premier enfant, se déplacera vers la droite pour couvrir le bord droit et vice-versa pour le bord gauche. Donc peu importe la vitesse de déplacement de notre caméra, nous aurons constamment un arrière-plan.

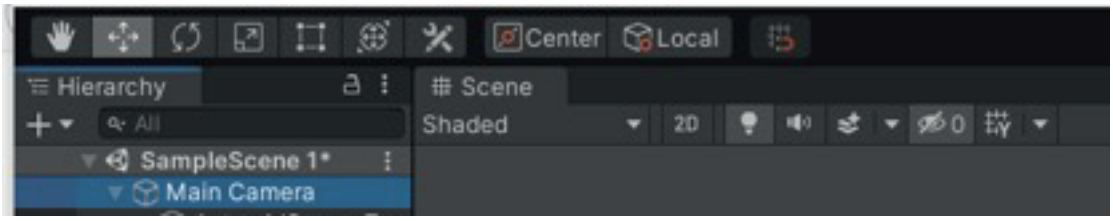
- **Copier-coller le script suivant (attention suite en page 4)**

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class backgroundLoop : MonoBehaviour{
   public GameObject[] levels;
   private Camera mainCamera;
   private Vector2 screenBounds;
   public float choke;
   public float scrollSpeed;

   private Vector3 lastScreenPosition;

   void Start(){
      mainCamera = gameObject.GetComponent<Camera>();
      screenBounds = mainCamera.ScreenToWorldPoint(new Vec-
tor3(Screen.width, Screen.height, mainCamera.transform.position.z));
      foreach(GameObject obj in levels){
         loadChildObjects(obj);
      }
      lastScreenPosition = transform.position;
   }
   void loadChildObjects(GameObject obj){
      float objectWidth = obj.GetComponent<SpriteRenderer>().bounds.size.x -
choke;
      int childsNeeded = (int)Mathf.Ceil(screenBounds.x * 2 / objectWidth);
      GameObject clone = Instantiate(obj) as GameObject;
      for(int i = 0; i <= childsNeeded; i++){
         GameObject c = Instantiate(clone) as GameObject;
         c.transform.SetParent(obj.transform);
         c.transform.position = new Vector3(objectWidth * i, obj.transform.posi-
tion.y, obj.transform.position.z);
         c.name = obj.name + i;
      }
      Destroy(clone);
      Destroy(obj.GetComponent<SpriteRenderer>());
   }
```

```
    void repositionChildObjects(GameObject obj){
        Transform[] children = obj.GetComponentsInChildren<Transform>();
        if(children.Length > 1){
            GameObject firstChild = children[1].gameObject;
            GameObject lastChild = children[children.Length - 1].gameObject;
            float halfObjectWidth = lastChild.GetComponent<SpriteRende-
rer>().bounds.extents.x - choke;
            if(transform.position.x + screenBounds.x > lastChild.transform.position.x
+ halfObjectWidth){
                firstChild.transform.SetAsLastSibling();
                firstChild.transform.position = new Vector3(lastChild.transform.posi-
tion.x + halfObjectWidth * 2, lastChild.transform.position.y, lastChild.trans-
form.position.z);
            }else if(transform.position.x - screenBounds.x < firstChild.transform.posi-
tion.x - halfObjectWidth){
                lastChild.transform.SetAsFirstSibling();
                lastChild.transform.position = new Vector3(firstChild.transform.posi-
tion.x - halfObjectWidth * 2, firstChild.transform.position.y, firstChild.trans-
form.position.z);
            }
        }
    }
    void Update() {
        Vector3 velocity = Vector3.zero;
        Vector3 desiredPosition = transform.position + new Vector3(scrollSpeed, 0,
0);
        Vector3 smoothPosition = Vector3.SmoothDamp(transform.position,
desiredPosition, ref velocity, 0.3f);
        transform.position = smoothPosition;
    }
    void LateUpdate(){
        foreach(GameObject obj in levels){
            repositionChildObjects(obj);
            float parallaxSpeed = 1 - Mathf.Clamp01(Mathf.Abs(transform.position.z /
obj.transform.position.z));
            float difference = transform.position.x - lastScreenPosition.x;
            obj.transform.Translate(Vector3.right * difference * parallaxSpeed);
        }
        lastScreenPosition = transform.position;
    }
}
```

4 Assigner le script à la main-caméra

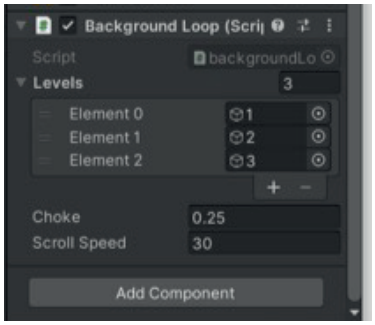
Vous pouvez enregistrer le script et l'assigné à votre «***main caméra»*** dans ***« Inspector_Add-Component_Script_backgroundLoop »*** ou tout simplement en en faisant un cliquer-glisser de la fenêtre ***«Project»*** à la **«main-caméra.»**



5 Rentrer les informations du script

Une fois votre script ajouté à la ***«main-caméra»,*** nous allons modifier, dans la fenêtre «***Inspector***», les informations du script. Donc dans « ***Size*** » il faut définir le nombre d'image que vous avez pour votre Background. Dans notre cas nous en avons 3 (le fond de la ville (1), les bâtiments-violet(2), et les bâtiments-noir(3)).

Vous pouvez ensuite assigner à chaque élément [Elément 0, Elément 1, Elément 2...] le game-objet correspondant à vos différentes images du background. **L'élément 0**, doit être l'élément le plus éloigné de la caméra (dans notre cas : le fond de la ville (1)) et **l'élément 2** le plus proche (dans notre cas : les bâtiments-noir(3). . Ajoutez ensuite la valeur 0,25 dans l'encadré « Choke », puis définissez la vitesse de votre Parallaxe effect, dans notre cas nous allons mettre « 30 ». Plus vous augmentez ce chiffre, plus la vitesse augmentera.



6 Lancez «play», et modifier la vitesse si besoin! C'est fini !

Tu peux maintenant créer la suite de ton jeu !