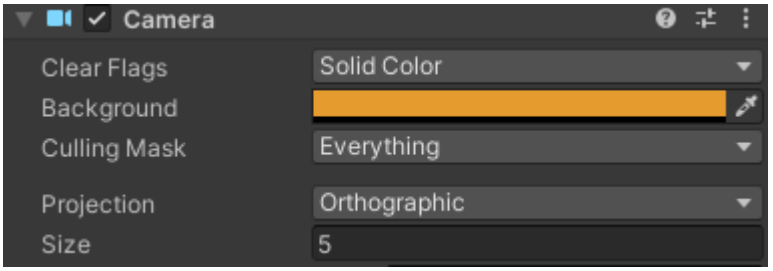
TP : Clicker

Ce TP guidé vous expliquera comment réaliser un clicker

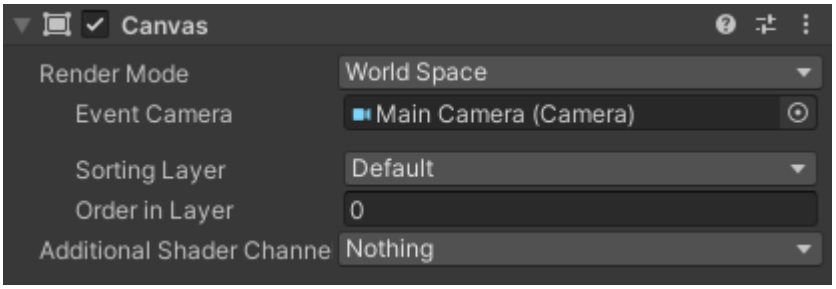
1. Vous trouverez les Assets nécessaires à la réalisation de ce TP à cette adresse :
   https://opengameart.org/content/svg-flat-clicker-rpg-kit
   https://opengameart.org/content/ui-pack-space-extension

   Vous pouvez si vous le souhaitez utiliser les assets de votre choix
2. Il s'agit d'un jeu en 2D pure : vérifiez que votre Camera est en orthographic

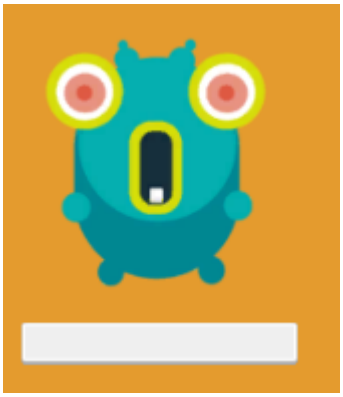


3. Positionnez un monstre sur la scene, étant donné qu'il s'agira d'un élément du gameplay affichez le en tant que SpriteRenderer

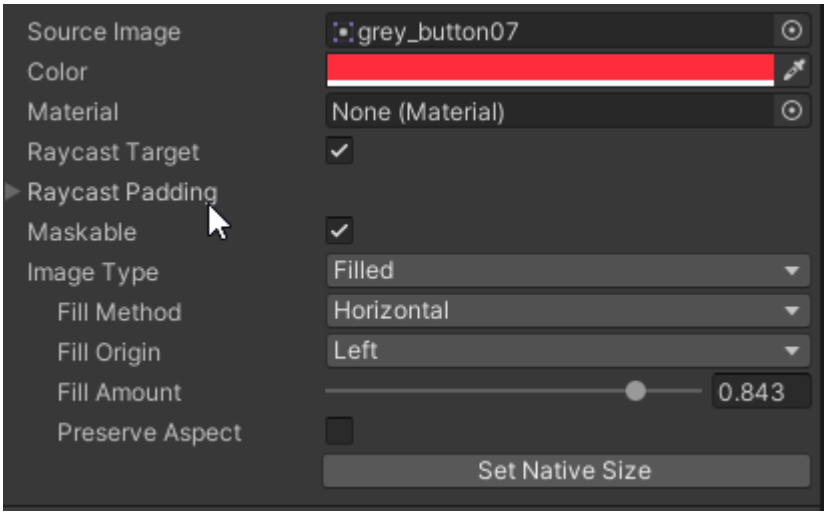
4. Ajoutez une barre de vie à ce monstre
    a. Il va falloir positionner l'UI afin d'ajouter du texte.
    Or le monstre est positionné dans le monde.
    Le plus simple est d'ajouter un Canvas qui sera affiché en WorldSpace afin de bien le positionner sous le monstre.

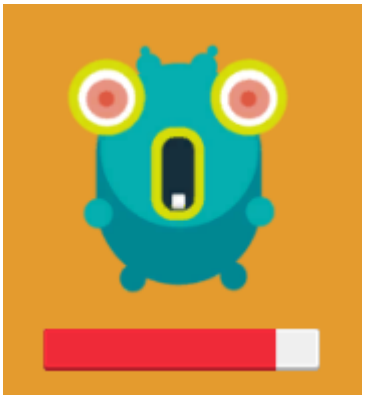


    b. Ajoutez-y une première image qui fera office de background de la barre de vie

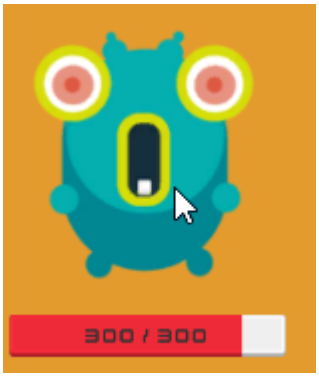


    c. Ajoutez une seconde image à la même position en imagetype "Filled" qui sera plus ou moins remplie en fonction de la vie du monstre.





d. Ajoutez un champ de texte afin d'afficher la vie du monstre, utilisez une autre police que celle par défaut.



5. Lorsque le joueur clique sur le monstre il faudra lui faire perdre des points de vie.
   a. Ajoutez un GameObject nommé "_Scripts" et ajoutez lui un script nommé "MainGame"
   b. Il faut vérifier si le joueur clique sur le monstre, pour cela ajoutez un BoxCollider2D au monstre en tant que trigger.
   Afin de déterminer si le curseur de la souris est sur le monstre ou non lors du clic il va falloir lancer un RayCast, il s'agit de lancer un rayon à partir de la position du curseur de la souris et de voir si ce rayon rencontre un collider.
   Attention cependant les coordonnées de la souris donné par la fonction Input.mousePosition sont en coordonnées écran (ils sont fonction de la résolution de l'écran), alors que la fonction Physics2D.Raycast attends une position world (en coordonnées dans le monde), il faudra donc dans un

premier temps convertir les coordonnées écran en coordonnées monde grâce à la fonction ScreenToWorldPoint de la classe Camera.
Utilisez ce code afin d'afficher le nom de l'élément sur lequel le joueur a cliqué afin de constater le bon fonctionnement

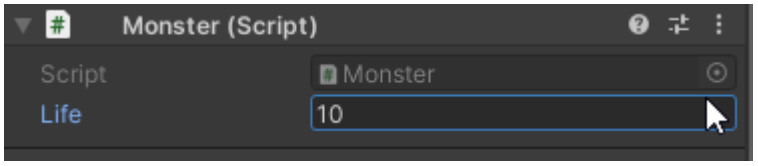
```
void Update()
{
    if (Input.GetMouseButtonDown(0))
    {
        Vector3 world = Camera.main.ScreenToWorldPoint(Input.mousePosition);
        RaycastHit2D hit = Physics2D.Raycast( world , Vector2.zero);

        if (hit.collider != null)
        {
            Debug.Log(hit.collider.name);
        }
    }
}
```

c. Il faut maintenant ajouter des points de vie au monstre, pour cela créez un nouveau script nommé "Monster".
Ajoutez une propriété publique nommée Life.

```
public class Monster : MonoBehaviour
{
    public int Life;
}
```

d. Ajoutez le component sur votre monstre et donnez lui 10 points de vie



e. Il suffit désormais de retirer des points de vie lors du clic sur le monstre.
Pour cela il faut récupérer le component Monster du collider qui a été détecté lors du Raycast.
et ensuite enlever 1 point de vie et l'afficher dans la console pour vérifier.

6. Affichez la vie sur l'ui :
   a. Dans votre classe Monster ajoutez une référence vers le component de Text qui contiendra la vie .

```
public Text TextLife;
```

   b. Ajoutez une référence vers l'image de la barre de vie dont on modifiera la taille en fonction de la vie
   c. Créez une fonction qui mettra à jour ce component Text en fonction de la vie

```
void UpdateLife()
{
    TextLife.text = Life.ToString();
}
```

d. Il faudrait aussi mettre à jour la taille de la barre de vie en fonction du pourcentage de vie, mais pour faire cela il faut garder en mémoire l'information de la vie maximale.
Ajoutez cette variable dans votre classe Monster.

```
int _lifeMax;
```

et mettez à jour votre fonction update life pour afficher la vie maximale à la fois sur le champ de texte et sur l'image.

```
void UpdateLife()
{
    TextLife.text = $"{Life}/{_lifeMax}";

    float percent = (float)Life / (float)_lifeMax;
    ImageLife.fillAmount = percent;
}
```

e. Il ne reste plus qu'à appeler la fonction lors de la création du Monster, tout en initialisant la valeur de _lifeMax

```
private void Awake()
{
    UpdateLife();
}
```

f. Il suffit maintenant d'appeler la fonction UpdateLife chaque fois que l'on modifie la vie du monstre, attention à ne pas oublier de passer la fonction UpdateLife en public

```
Monster monster = hit.collider.GetComponent<Monster>();
monster.Life--;
monster.UpdateLife();
```

g. Attention par la suite si l'on modifie la valeur de la variable Life la fonction UpdateLife ne sera pas appelée, cela pourrait conduire à différents bugs. Afin d'empêcher ce potentiel problème nous allons procéder à un certain nombre de changements :
   i. Passage de la fonction UpdateLife en privée : il n'y aura plus d'intérêt à appeler la fonction puisque cela sera fait automatiquement lorsque la valeur de la variable Life sera modifiée
   ii. Ajout d'une méthode Hit qui enlève de la vie et qui met à jour l'affichage de celle-ci

```
public void Hit(int damage)
{
    Life -= damage;
    UpdateLife();
}
```

   iii. Il ne faut plus que la variable Life soit public afin de ne pas être modifiée de l'extérieur. Mais l'on souhaite toujours pouvoir modifier sa valeur dans Unity, pour cela on utilise l'attribut [SerializeField]

```
[SerializeField]int _life;
```

(utilisez la touche F2 pour renommer la variable afin de renommer en une fois toutes les occurrences)

iv. Il ne reste plus qu'à remplacer l'appel de fonction.

```
Monster monster = hit.collider.GetComponent<Monster>();
monster.Hit(1);
```

7. Lors du click sur le monstre il manque un feedback de hit.
   Importez la librairie DoTween dans votre packageManager.
   Ajoutez le code suivant lors du hit afin d'ajouter le feedback.

```
transform.DOPunchScale(new Vector3(0.2f, 0.2f, 0), 0.3f);
```

   Un premier problème survient : la barre de vie est aussi impactée par le scale car elle est enfant du gameobject Monster.

   a. Ajoutez un nouveau gameObject enfant de monster appelé Visual qui contiendra uniquement le visuel du monstre, vous pouvez donc supprimez le visuel sur le gameobject monster (le component SpriteRenderer)
   b. Ajoutez une référence vers le gameObject Visual dans votre code

```
public GameObject Visual;
```

   c. Effectuez le tween uniquement sur ce gameobject

```
Visual.transform.DOPunchScale(new Vector3(0.2f, 0.2f, 0), 0.3f);
```

   d. Le second problème est que si l'utilisateur clic très vite, un nouveau tween se lance en prenant comme taille référence la taille courante et donc potentiellement une taille plus petite, ce qui réduit petit à petit la taille de notre monstre.
      Afin d'éviter cela, il faut forcer le tween en cours à se finir avant de démarrer le nouveau.

```
Visual.transform.DOComplete();
Visual.transform.DOPunchScale(new Vector3(0.2f, 0.2f, 0), 0.3f);
```

8. Une fois le monstre tué il faudra être en mesure de laisser la place à un autre monstre.
   Pour cela il va falloir stocker les informations relative à un monstre :
   Un monstre est représenté par :
   son nom (string)
   sa vie (int)
   son image (sprite)
   Créez la classe MonsterInfos permettant de stocker ces informations (cette classe ne sera pas un component il ne faut donc pas la faire hériter de MonoBehavior.)

```
public class MonsterInfos
{
    public string Name;
    public int Life;
    public Sprite Sprite;
}
```

9. Dans la classe monster ajoutez une fonction SetMonster afin de mettre à jour le visuel et les informations du monstre en fonction du paramètre MonsterInfos.

```
public void SetMonster( MonsterInfos infos )
{
    _lifeMax = infos.Life;
    _life = _lifeMax;

    Visual.GetComponent<SpriteRenderer>().sprite = infos.Sprite;
}
```

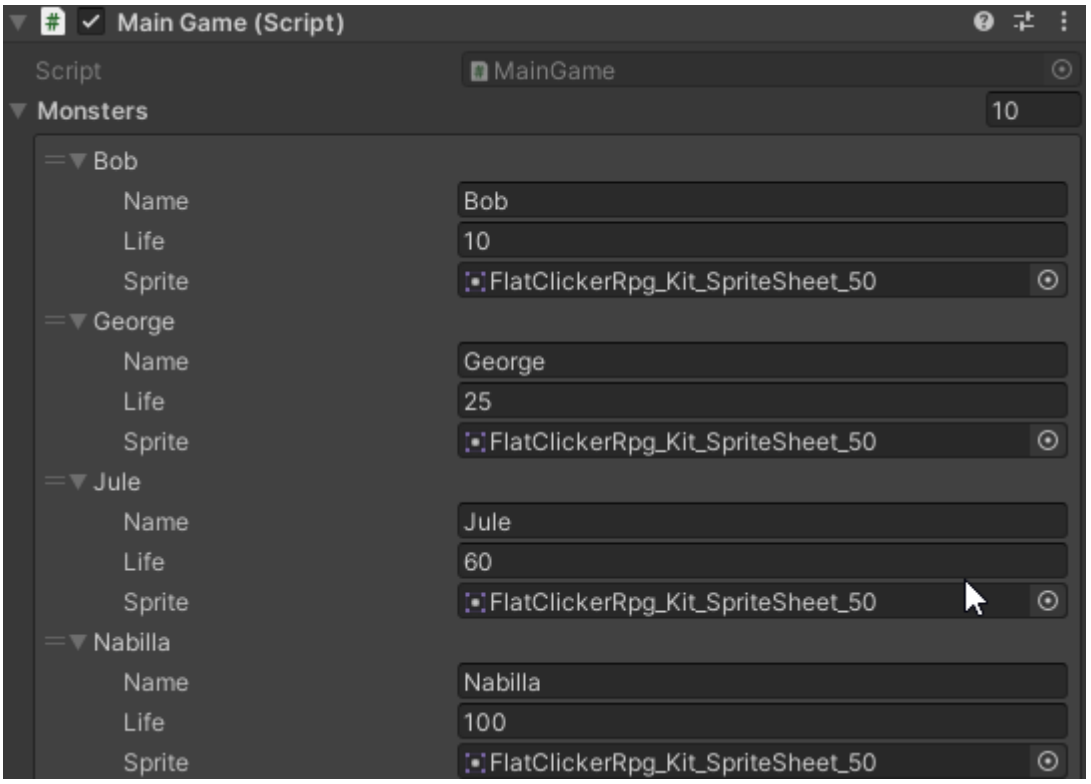
10. Ajoutez maintenant votre liste de MonsterInfos dans votre MainGame.cs

```
public List<MonsterInfos> Monsters;
```

Vous pouvez constater que votre liste n'apparaît pas dans l'inspecteur.
Pour cela il faut préciser que votre classe est sérialisable (c'est à dire qu'elle peut être enregistrée par Unity)

```
[Serializable]
public class MonsterInfos
{
```

Vous pouvez maintenant ajouter 10 monstres de difficultés croissantes



11. Il faut maintenant garder en mémoire quel est le monstre actuel, pour cela il suffit simplement de stocker l'index du monstre courant, cela permettra de connaître l'index du prochain monstre.

```
int _currentMonster;
```

12. Ici nous n'allons pas détruire le gameobject du monstre pour chaque monstre mais simplement recharger les informations, il faut donc une référence vers le component Monster à partir du MainGame.

```
public Monster Monster;
```

Il est donc maintenant possible de prendre le premier monstre de la liste et de l'affecter au lancement du jeu.

```
void Start()
{
    Monster.SetMonster(Monsters[_currentMonster]);
}
```

13. Enfin il est possible de passer au monstre suivant lorsque le monstre est tué :

```
public bool IsAlive()
{
    return _life > 0;
}
```

```
Monster monster = hit.collider.GetComponent<Monster>();
monster.Hit(1);

if ( monster.IsAlive() == false )
{
    NextMonster();
}
```

```
private void NextMonster()
{
    _currentMonster++;
    Monster.SetMonster(Monsters[_currentMonster]);

}
```

14. Lorsque le monstre perd de la vie, un feedback indiquant le nombre de points de vie perdu serait intéressant.
    a. Ajoutez un GameObject “FeedbackHitPoint” contenant un component Text dans le canvas du monstre

    b. Il pourra y en avoir plusieurs en même temps, il va donc falloir en faire un prefab et les créer dynamiquement.
    Mettez votre prefab dans un dossier nommé “Prefabs” pour plus de clarté et supprimez le gameobject de votre scène
    c. Ajoutez une référence vers le gameObject dans votre script

```
public GameObject PrefabHitPoint;
```

    d. Ajoutez une référence sur votre monster pour récupérer son canvas

```
public Canvas Canvas;
```

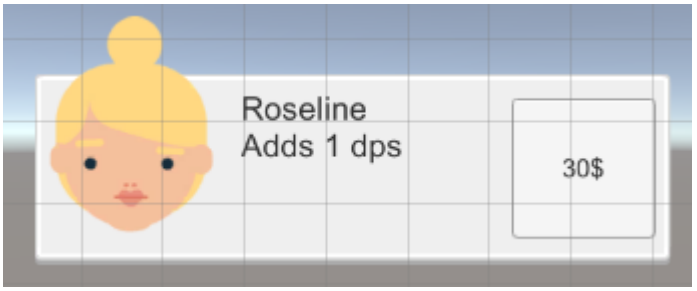
    e. Il faut désormais instancier votre prefab en tant qu’enfant du canvas, à la bonne position et avec une petite animation, il faut aussi supprimer ce gameobject une fois son animation terminée.

```
monster.Hit(1);
GameObject go = GameObject.Instantiate(PrefabHitPoint, monster.Canvas.transform, false);
go.transform.localPosition = new Vector3(0, 0, 0);
go.transform.DOLocalMoveY(150, 0.8f);
go.GetComponent<Text>().DOFade(0, 0.8f);
GameObject.Destroy(go, 0.8f);
```

    f. Enfin ajoutez un peu d’aléatoire sur la position initiale du feedback

```
go.transform.localPosition = UnityEngine.Random.insideUnitCircle * 100;
```

15. Désormais il va falloir placer les différentes améliorations déblocables.
    Afin de pouvoir en afficher un grand nombre, celles-ci seront positionnées dans une scrollview.
    Ajoutez une ScrollView dans un nouveau Canvas.
16. Créez un gameObject UpgradeUI contenant tous les éléments suivants :
    un background, un texte, une image et un bouton



    Et transformez le en prefab
17. Afin de pouvoir modifier les champs de UpgradeUI, créez un script nommé UpgradeUI et assignez-le à votre prefab éponyme.
    Ajoutez une référence vers l'image, le texte et le texte du prix

```
public class UpgradeUI : MonoBehaviour
{
    public Image Image;
    public Text Text;
    public Text TextCost;
}
```

18. Il faut maintenant Définir les données des différentes upgrades
    a. Créez une nouvelle classe nommé Upgrade et qui n'hérite pas de MonoBehavior

```
public class Upgrade
{

}
```

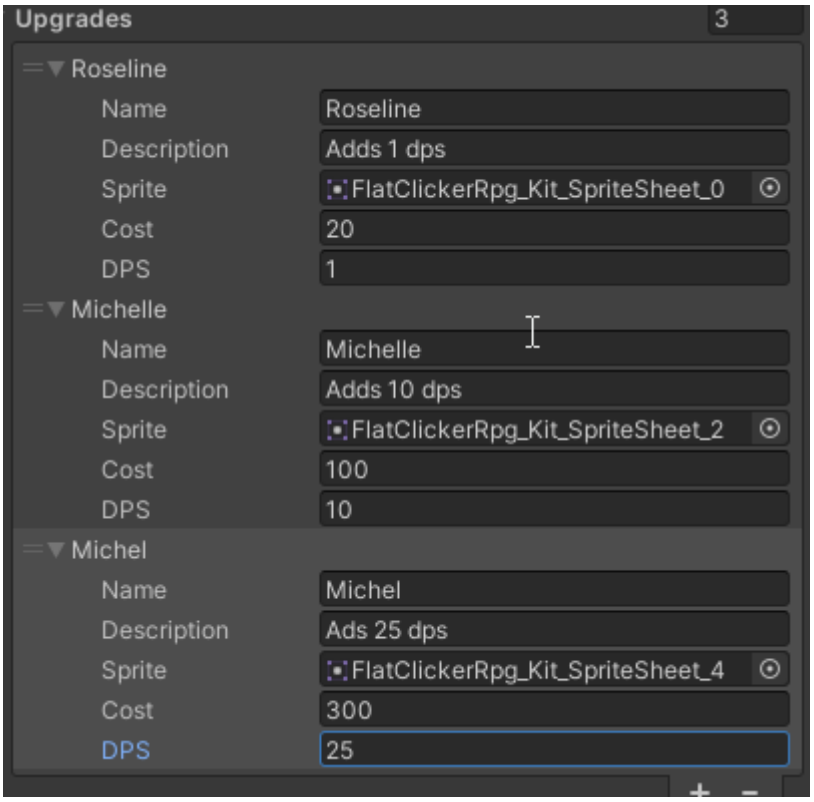
    b. Ajoutez les différents champs de l'upgrade :

```
public class Upgrade
{
    public string Name;
    public string Description;
    public Sprite Sprite;
    public int Cost;
    public int DPS;
}
```

    c. Ajoutez une liste d'Upgrade dans MainGame

```
public List<Upgrade> Upgrades;
```

d. Ajoutez dans cette liste les différentes upgrades dans Unity



Si vous ne voyez pas cette liste c'est qu'il vous manque l'attribut [Serializable] devant votre classe Upgrade

```
[Serializable]
public class Upgrade
```

e. Il va maintenant falloir instancier les prefabs UpgradeUI pour chaque Upgrade dans la liste
Ajoutez une référence vers le prefab dans votre MainGame

```
public GameObject PrefabUpgradeUI;
```

f. Les prefabs seront instanciés dans le gameobject Content de la scrollview : ajoutez une référence vers ce gameobject

```
public GameObject ParentUpgrades;
```

g. Dans le Start vous pouvez parcourir chaque Upgrade et instancier un UpgradeUI pour chacun

```
foreach (var upgrade in Upgrades)
{
    GameObject go = GameObject.Instantiate(PrefabUpgradeUI, ParentUpgrades.transform, false);
    go.transform.localPosition = Vector3.zero;
}
```

Le problème c'est que ces UI ne correspondent pas aux upgrade.

h. Pour corriger cela il est possible de créer une fonction pour initialiser une UpgradeUI en lui donnant une upgrade associée.
Au moment de cette initialisation il serait intéressant de stocker l'upgrade

associée afin de pouvoir la réutiliser plus tard lors du clic par exemple.

```
public void Initialize( Upgrade upgrade )
{
    _upgrade = upgrade;
    Image.sprite = upgrade.Sprite;
    Text.text = upgrade.Name + System.Environment.NewLine +  upgrade.Description;
    TextCost.text = upgrade.Cost + "$";
}
```

i. Il suffit maintenant d'appeler l'initialize pour chaque UpgradeUI instancié:

```
go.GetComponent<UpgradeUI>().Initialize(upgrade);
```

j. Vous obtenir ceci :

19. Il faut maintenant faire en sorte que lorsque le joueur débloque une upgrade, celle-ci est bien effective : si j'achète Roseline j'ai un dps automatique de 1 dégât par seconde sur le monstre. L'achat de cet upgrade se fera dans un second temps.

a. Dans MainGame ajoutez une liste qui contiendra les upgrade débloqués

```
List<Upgrade> _unlockedUpgrades = new List<Upgrade>();
```

b. Dans l'update effectuez les dommages de toutes les upgrades débloqués, il faut mettre en place un timer pour effectuer les dommages toutes les secondes

```
_timerAutoDamage += Time.deltaTime;

if (_timerAutoDamage >= 1.0f)
{
    _timerAutoDamage = 0;
    foreach (var upgrade in _unlockedUpgrades)
    {
        Monster.Hit(upgrade.DPS);
    }
}
```

c. Il serait bien de mettre en place les feedbacks de la même manière que lorsque l'on clique sur le monstre, finalement ce code sera commun à celui du clic, il est donc préférable de créer une nouvelle fonction Hit au sein de la classe Maingame et de l'appeler dans les 2 cas

```
void Hit(int damage,Monster monster)
{
    monster.Hit(damage);

    GameObject go = GameObject.Instantiate(PrefabHitPoint, monster.Canvas.transform, false);
    go.transform.localPosition = UnityEngine.Random.insideUnitCircle * 100;
    go.transform.DOLocalMoveY(150, 0.8f);
    go.GetComponent<Text>().DOFade(0, 0.8f);
    GameObject.Destroy(go, 0.8f);

    if (monster.IsAlive() == false)
    {
        NextMonster();
    }
}
```

d. Il faut maintenant faire en sorte d'ajouter l'upgrade à la liste des upgrades débloqués lors du clic sur le bouton.
Ajoutez une fonction AddUpgrade dans MainGame

```
public void AddUpgrade( Upgrade upgrade )
{
    _unlockedUpgrades.Add(upgrade);
}
```

e. Ajoutez une fonction OnClick dans UpgradeUI

```
public void OnClick()
{

}
```

f. Il faut appeler le AddUpgrade de Maingame à partir du OnClick de UpgradeUI.
Il y a de nombreuses façons de faire plus ou moins propres.
L'une des moins sales consiste à ajouter un singleton sur MainGame (puisque qu'il ne peut y avoir qu'une seule instance de MainGame)

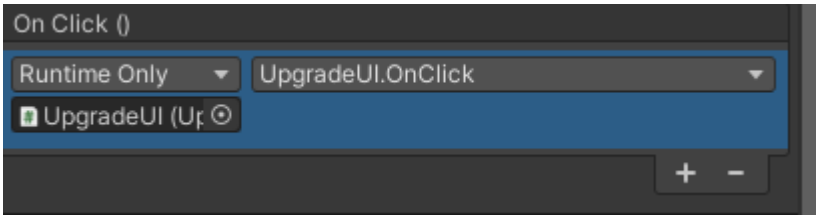
```
public static MainGame Instance;

private void Awake()
{
    Instance = this;
}
```

g. Désormais il est possible d'accéder à MainGame de n'importe où dans le code

```
public void OnClick()
{
    MainGame.Instance.AddUpgrade(_upgrade);
}
```

h. N'oubliez pas de liez la méthode OnClick au bouton associé



20. Vous pouvez désormais tester votre jeu : cela fonctionne mais les feedbacks n'affichent pas la bonne valeur : corrigez cela !

Pour aller plus loin :

- Le système d'achat des upgrades n'est pas en place : ajoutez un sytème d'or :
  Lors de chaque Hit le joueur gagne de l'or et cet or est affiché dans l'interface.
  Si le joueur n'a pas assez d'or, il ne peut pas acheter les upgrades.
  Lors de l'achat d'une upgrade le bouton d'achat disparaît afin de ne pas pouvoir l'acheter 2 fois.
- Ajoutez des upgrades qui permettent d'améliorer le clic des joueurs
- Faites en sorte que chaque Upgrade ait plusieurs niveau d'upgrades