Diablo-like (TP2)

Programmation et intégration en jeu vidéo (NWE.29)

420-JWJ-BT

Développement de mécanique de jeu

Professeur: **Étienne Bédard**
Contact : Par Mio sur Omnivox
Local: **A-32**

Énoncé**: Date 2019-10-09**
Remise: **Date Vendredi 15 Novembre 23h59**

Contexte de réalisation

Votre mandat est de créer un “donjon” à la diablo. C’est à dire, un personnage se déplaçant avec le clic gauche de la souris, il devra pouvoir attaquer à distance à l’aide du même clic. Si le joueur clique sur un “ennemie” ou un objet de type “destructible” le personnage doit tirer, sinon le personnage se déplace dans la direction du clic. La caméra devra suivre le personnage de façon à ce qu’il soit centré dans l’écran sans toutefois dépasser les limites du niveau.

Votre jeu devra avoir au minimum un élément interactif supportant le gameplay de votre donjon, par exemple: un levier qui active une porte, un puzzle où le joueur doit déplacer des objets pour se faire un chemin, etc. Vous devrez faire valider votre élément avec moi d’abord. Chaque élément visuel du

jeu devra être clairement identifié, par exemple: tout ce qui peut tuer le joueur est rouge, tous les éléments interactifs sont des sphères, etc. Votre code devra suivre le standard de programmation décrit à la fin de ce document.

Avant propos

Ce travail pratique porte sur les déplacements par le raycasts, la caméra et les interactions. Il vous sera aussi demander de programmer des interactions entre les objets.

Objectifs

- Mécanique de tir (PlayerController). **3 points**
- Mécanique de déplacement (PlayerController). **3 points**
- Mécanique de suivi caméra (CameraFollow). **3 points**
- Mécanique interactive de l'environnement. **3 points**
- Identification visuelle claire dans le niveau. **2 points**
- Respect du standard de programmation. **4 points**
- Début du niveau et fin du niveau. **2 points**
- Possibilité d'échouer et de recommencer à même le niveau. **2 points**

Procédure de remise

Le tout devra être remis au plus tard le Vendredi le 15 novembre avant l'heure fatidique de 23h59.

On doit remettre le projet dans un dossier compressé .ZIP, .RAR ou .7Z Portant vos noms (ex. BillyBobThorthon.zip) sur dépôt dans le dossier prévu à cet effet.

N'hésitez pas à entrer en contact avec le professeur pour des informations à éclaircir.

Bon travail!

- Étienne Bédard

Standard de programmation:

- Vos objets, prefabs et scripts doivent avoir un nom en relation directe avec leur utilisation, par exemple, pour un script de piège MovingTrap.cs, pour un cube représentant un boite destructible DestructibleCrate, et etc.

- Vos prefabs doivent être dans un dossier Prefab, vos scripts dans un dossier Script et etc.

- Vos fonctions doivent tous avoir un accesseur(private ou public) et commencer par une majuscule, par exemple: public void ShootTarget()

- Vos variables membres (variable ne faisant pas partie d'une fonction) doivent commencer par m_ et chaque "mot" doit être séparé par une majuscule, par exemple: public bool m_RightWayToDeclareAVariable = true;

- Les paramètres(ou argument) de fonction doivent commencer par "i_",par exemple: public void OnTriggerEnter(Collider i_Col)

- Votre indexation et "scope" doit respecter les "Tab"(voir exemple de code respectant tous les standard demander)

Exemple de scope et indexation

Une bracket ouverte "{"signifie le début d'un scope et une bracket fermer "}" indique la fin d'un scope.
Exemple:

```
public class Scope : MonoBehaviour
{
    tout ce qui se trouve à l'intérieur à une tabulation (TAB)
    de cette façon la lecture est plus facile.
}
```

Exemple de plusieurs Scope:

```
public class MultipleScope : MonoBehaviour
{
    private void Start()
    {
        if(true)
        {
            Tabulation;
        }
    }
}
```

Remarquer que les Bracket ouverte "{" vont en "échelle de plus en plus loin vers la droite, tandis que les Bracket fermés nous ramènent au début vers la gauche, indiquant clairement un début et une fin de class, condition, fonction et etc.

Exemple de code

```
public class Jumper : MonoBehaviour
{
	private bool m_CanJump;
	private Rigidbody m_RigidBody;
	private int m_CurrentJumpIteration;

	private void Start ()
	{
		m_CanJump = true;
		m_RigidBody = GetComponent<Rigidbody>();
		m_CurrentJumpIteration = 0;
	}

	private void Update ()
	{
		if(m_CanJump && Input.GetKeyDown(KeyCode.Space))
		{
			Jump();
		}
	}

	private void Jump()
	{
		if(m_CurrentJumpIteration >= 2)
		{
			m_CanJump = false;
		}
		else
		{
			m_RigidBody.AddForce(transform.up * 250f);
			m_CurrentJumpIteration += 1;
		}
	}

	private void OnTriggerEnter(Collider i_Col)
	{
		if(i_Col.gameObject.tag == "Ground")
		{
			m_CanJump = true;
			m_CurrentJumpIteration = 0;
		}
	}
}
```