

Entity-Component-System

Bonjour à tous et bienvenue dans ce workshop où nous allons créer un prototype d'ECS pour comprendre mieux son fonctionnement et son implémentation.

1.Qu'est-ce que c'est ?

L'ECS est un modèle d'architecture de code souvent utilisé dans la création de jeu vidéo ou de game engine. Ce modèle a vu le jour pour proposer une alternative, plus facile maintenable, au modèle de hiérarchie en héritage déjà beaucoup présent dans le milieu des jeux vidéo.

La complexité de l'ECS et pourquoi j'ai proposé l'ECS en sujet est qu'il n'existe pas de base universelle pour créer un ECS, chaque partie est plus ou moins libre a interprétation et peut changer du tout au tout quant à l'implémentation d'un ECS à un autre.

Ce modèle se base sur trois grands piliers que l'on va détailler de ce pas.

2.La structure de l'ECS

Entity : Une entité est une représentation de tous les éléments du programme. Dans l'exemple d'un jeu vidéo ces concepts sont des « entités » : joueur, arme, balle, caméra, texte, scénario, etc.... Ainsi donc pour différencier une balle d'un joueur qui sont tous les deux des entités, nous allons leur attribuer des comportements sous la forme de composants détachables.

Component : Un component est un comportement que l'on peut attacher à une entité aussi bien à la compilation qu'au runtime. Par exemple, ces concepts sont des composants : vitesse, physique, possibilité d'émettre de la lumière, etc...

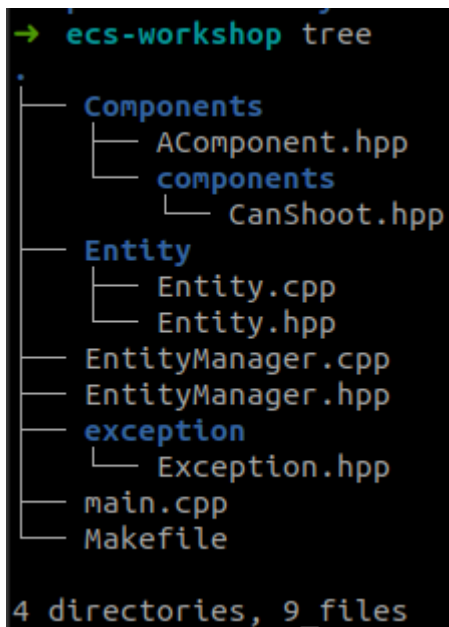
System : Un system et une méthode qui va manipuler des entités et/ou des composants pour les mettre à jour, modifier, supprimer. Ils représentent les intermédiaires entre l'ECS et le reste du programme.

3.Créons un ECS

Il est temps de coder, et dans ce workshop je vais vous proposer de le faire en C++. Même si l'ECS est né dans le but de se séparer de l'héritage, il l'utilise d'une autre manière et donc il est fortement conseillé pour implémenter un ECS de le faire dans un langage objet comme java ou C++.

Comme recréer un ECS n'est pas une tâche, je vous propose de démarrer d'une base fait maison avec quelques bouts de code manquants qu'il faudra que vous complétiez.

Voici le lien du repository : <https://github.com/Preant/ecs-workshop>



```
→ ecs-workshop tree
├── Components
│   ├── AComponent.hpp
│   └── components
│       └── CanShoot.hpp
├── Entity
│   ├── Entity.cpp
│   └── Entity.hpp
├── EntityManager.cpp
├── EntityManager.hpp
├── exception
│   └── Exception.hpp
├── main.cpp
└── Makefile

4 directories, 9 files
```

Comme le but n'est pas de vous faire écrire des tests, je vous ai mis à disposition un main de test ainsi qu'un makefile pour le compiler, c'est grâce à eux que vous pourrez valider vos exercices. Pour tester c'est très simple tapez make dans le terminal puis lancez le binaire ecs-test avec un premier argument le numéro de l'exercice auquel vous êtes. Pour valider, tous les exercices doivent être réussis même ceux des précédentes étapes.

```
→ ecs-workshop make
[ OK ] Compiling main.cpp
[ OK ] Compiling EntityManager.cpp
[ OK ] Compiling Entity/Entity.cpp
[ FILES COMPILED ] 3
Binary : ecs-test created sucesfully.
→ ecs-workshop ./ecs-test 1

Test01:createSingleCmpt
CanShoot not created
one or several tests has failed

TESTS FAILED
→ ecs-workshop ./ecs-test 4

Test01:createSingleCmpt
CanShoot not created

Test02:createEmptyEntity
error while creating a new entity.

Test03:addCmpttoEmptyEntity
error while creating a new entity.

Test07:removeEntity
error while creating a new entity.
one or several tests has failed

TESTS FAILED
```

3.1Components

Tout d'abord, avant de nous attaquer à l'architecture de cet ECS nous allons créer un component représenter un comportement. Ce component est défini dans CanShoot.hpp et comme vous pouvez le voir, il est assez... inutile. Le but sera juste de le faire interagir avec des entités.

Exo1 :

Dans main.cpp, jetez un œil à la fonction « TcreateSingleCmpt ». On va démarrer avec un exercice simple, instanciez un ecs ::Cmpt qui représentera notre comportement CanShoot.

3.2Entités

Passons maintenant au cœur de l'ECS, les entités. Elles sont définies dans le header Entity.hpp et sont créées par la classe EntityManager. Nous allons essayer de créer une entité vide, c'est-à-dire sans aucun comportement.

Exo2 :

Dans EntityManager.cpp, jetez un œil à la fonction create. Votre travail est de créer une ecs ::Entt(cf. constructeur de la classe Entity) ainsi qu'un nouvel Id puis de l'ajouter à la std ::map contenue dans l'EntityManager. Soyez vigilants sur la valeur que renvoie cette fonction, il s'agit d'un itérateur sur une paire de la map !

Désormais vous êtes capable de créer des entités mais le plus intéressant est à venir, nous allons ajouter notre comportement CanShoot à notre entité nouvellement créée.

Exo 3 :

Dans Entity.cpp, jetez un œil à la fonction addBehaviour. Votre rôle ici est de rajouter au vecteur de component, le component passé en paramètre et comme par magie, votre entité sera capable d'utiliser les méthodes mises a disposition dans ce component.

3.3Entitymanager

L'EntityManager est une classe (qui peut être un singleton) qui va contenir toutes nos entités ainsi que nos systèmes. Vous avez déjà pu constater l'importance de cette classe dans la création d'entités et nous allons continuer à l' étoffer avec la méthode qui nous permet de supprimer des entités.

Exo 4 :

Dans EntityManager.cpp, jetez un œil à la fonction remove. Vous devez supprimer dans la map la paire qui fait référence à l'Id passé en paramètre. Soyez vigilants à ce que l'Id soit bien attribué avant de procéder à une quelconque suppression (cf. ecs ::EntityManager ::isIdGiven).

Parfait ! Vous avez officiellement manipulé un ECS. Cependant, ne nous enflammons pas, cet ECS n'est qu'une coquille vide pas très pratique, je m'explique.

Ce qui va prendre le plus de place dans votre ECS, c'est bel est bien les composants car c'est eux qui contiendront toute la logique de vos entités, ainsi, vous aurez probablement une liste d'une cinquantaine de composants à la fin de votre projet. De plus, il est possible d'étoffer votre ECS de manière à rendre plus simple la manipulation de tous les éléments du modèle.

Voici quelques pistes d'améliorations pour vous aider à manipuler votre ECS :

-mise en place de blueprint et d'une factory pour créer des entités déjà dotées de comportements

-mise en place de design pattern pour accélérer ou sécuriser votre ECS (visiteur, singleton, stratégie, etc...)

-mise en place d'un système d'observateur qui permettrait à vos composants d'écouter les événements d'autres composants (ex : Dans une arme votre component chargeur va écouter le component CanShoot pour automatiquement mettre à jour le nombre de balles)

- mise en place de dépendances entre les composants au moment de leur création (ex : un Component CanShoot ne peut pas être appliqué sur une entité n'ayant pas de component Ammo)