# ECS Documentation

For this project, we made an ecs-architecture implementation. It is then possible to develop your own games with our code and even to use a different graphical library.

First of all, you need to declare a SceneManager to manage your games' scenes. Create your scenes in a dedicated function that will set the SceneManager's variable _current to your own scene. Finally, add your scene creation function in loadScene. When called, your scene will then be set on _current.

```
71  void SceneManager::createGame()
72  {
73      _current = std::make_shared<GameScene>(_ecs);
74  }
75
76  void SceneManager::createChooseRoom()
77  {
78      _current = std::make_shared<ChooseRoomScene>(_ecs);
79  }
80
81  void SceneManager::loadScene(const std::string& name)
82  {
83      if (name == "Game")
84          createGame();
85      else if (name == "Menu")
86          createMenu();
87      else if (name == "ChooseRoomSystem")
88          createChooseRoom();
89      else
90          std::cerr << "Unable to create scene : " << name << std::endl;
91  }
92
```

## Creating entities :

Before adding your *entities* and *systems* to your scene, you need to create them. For that, create a new folder inside **src/Entities/EntityConstructor/YourEntity** then inside of it, create a class that inherits from IEntityConstructor.

It's then inside YourEntity.cpp that you will add the entity's components. Call them in the function : `std::shared_ptr<ecs::entities::Entity> TestEntity::create(...)`

Declare them this way :

```
componentsManager->addPhysicComponent(std::make_shared<ecs::components:
:Position>(0, 0), toCreate);
```

YourEntity will now have a component position. You can create your personalized components also, but we will see that later.
Don't forget to give your entity a name in the getName()

## Creating systems:

After you created your first entity, you will need a *system*. For that, create a new folder inside **src/Systems/SystemConstructor/YourSystem** then inside of it, create a class that inherits from ASystem.

Your system constructor should look like this

```
YourSystem::YourSystem(std::shared_ptr<IManagerWrapper>
&managerWrapper, std::shared_ptr<ecs::entities::IEntityFactory>
&entityFactory, std::list<int> &entitiesToDelete) :
ASystem(managerWrapper, entityFactory, entitiesToDelete),
_elapsedTime(0)
```

Then simply add your system logic inside update().

## Back to your scene!

It is then inside your *scene constructor* that you will declare the *entities* and *systems* you will be using.

add a *system* like this :

```
_ecs->getSystemManager()->addSystem(std::make_shared<system::YourSystem
>(_ecs->getManagerWrapper(), _ecs->getEntityFactory(),
_ecs->getSystemManager()->getEntitiesToDelete()));
```

add an *entity* like this :

```
_ecs->getEntityFactory()->createEntity("yourEntityName");
```

/!\ It is also super important to add your entity to the entitiesFactory. Otherwise, the ECS won't find it. In order to do that, simply write this line below inside the SceneManager constructor!

```
_ecs->getEntityFactory()->addEntityConstructor(std::make_shared<entitie
s::YourEntity>());
```

/!\ Don't forget to add your files in the CMakeLists.txt

After these steps, our *_entityManager* will handle your entity on his own.

You can look for entities of a specific type like this :

```
 for (auto &it : _managerWrapper->getEntityManager()->getAllEntities())
{
std::shared_ptr<ecs::components::Position> pos =
std::dynamic_pointer_cast<ecs::components::Position>(_managerWrapper->g
etComponentManager()->getPhysicComponentOfSpecifiedType(it->getID(),
std::type_index(typeid(ecs::components::Position))));
```

and delete them easily :

```
_entitiesToDelete.push_front(it->getID());
}
```

## Adding resources :

If you didn't change the graphics library, use the ResourceManager constructor and load your image this way, it will create a Texture that the component Sprite is using.

```
loadTexture("yourResourceTextureName","./resources/yourResource.png");
```

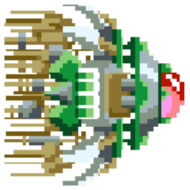
## Easy Start :

In order to start your project, you can use our pre-made components :

- ➔ Position
- ➔ Velocity
- ➔ Collision
- ➔ Rotation
- ➔ Damage
- ➔ Health
- ➔ Timer
- ➔ Animator
- ➔ Sprite
- ➔ Text
- ➔ Parallax
- ➔ Sound
- ➔ and many others ...

# Enemies: Types and Patterns

Enemy n°1 → small enemy that doesn't shoot and moves in a wave pattern
50% chance to appear



Enemy n°2 → a large and slow enemy that shoots and moves in a straight line
10% chance to appear

Enemy n°3 → leaping enemy that shoots and moves in a jumping pattern
20% chance to appear

Enemy n°4 → fast enemy that doesn't shoot but moves towards the player
20% chance to appear