HAI819I - Moteur de jeux
Documentation

Ingo Diab
Yahnis Saint-Val
Arthur Villarroya-Palau

2023

Table des matières

1 Avant-Propos

Le moteur étant trop lourd pour être mis sur Moodle, voici le lien github du projet : `https://github.com/IngoDiab/Projet_MoteurDeJeu.git`

La vidéo étant aussi trop lourde pour Moodle, voici le lien où la retrouver : `https://www.youtube.com/watch?v=iS5RG8KS6-Y`

2 Scènes

2.1 Création

Pour créer une nouvelle scène, il faut créer un script héritant de la classe Scène. Cette classe Scène est une classe abstraite contenant une méthode abstraite LoadScene() et deux méthodes d'update qui peuvent être surchargées Update() et LateUpdate().

```cpp
#include "engine/Scenes/Scene/Scene.h"

class Ground_Player;
class Landscape;
class Spaceship;
class Trophy;
class DirectionalLight;

class Scene_Earth final : public Scene
{
    bool mCollectibleAvailable = true;
    Ground_Player* mCharacter = nullptr;
    Landscape* mLandscape = nullptr;
    Spaceship* mSpaceship = nullptr;

    DirectionalLight* mSunLight = nullptr;

public:
    virtual void LoadScene() override;

private :
    Ground_Player* CreateCharacter();
    Spaceship* CreateSpaceship();
    Landscape* CreateLandscape();
    Trophy* CreateTrophy();
    void InitEditorCamera() const;
    void BindCharacterInput();
};
```

FIGURE 1 – Exemple de Scène

2.2 Initialisation

La méthode LoadScene() correspond à l'initialisation de la scène, elle est appelée par le SceneManager lorsque l'on demande le load de la scène. Lors du load d'une scène, la scène actuelle est unload (les objets ayant été créés dans la scène ayant comme durabilité SCENE et non PERSISTENT seront détruit et les inputs associés seront supprimés). Ces suppressions sont directement gérées dans la classe Scene donc il n'est pas utile, dans ce cas, de surcharger la méthode.

```
void Scene_Earth::LoadScene()
{
    Skybox* _skybox = Skybox::Instance();
    _skybox->ChangeSkybox("Textures/Skybox/Base/", ".jpg");

    mCharacter = CreateCharacter();
    mSpaceship = CreateSpaceship();
    mLandscape = CreateLandscape();

    InitEditorCamera();
    BindCharacterInput();

    mCharacter->ClipToLandscape(mLandscape);
    mSpaceship->ClipToLandscape(mLandscape);

    ObjectManager* _objectManager = ObjectManager::Instance();

    DirectionalLight* _sunLight = _objectManager->Create<DirectionalLight>();
    _sunLight->SetColor(vec3(1));
    _sunLight->SetDirection(vec3(1,-1,1));

    CreateTrophy();
}
```

FIGURE 2 – Exemple d'initialisation de la Scène

2.3 Instanciation

Pour instancier une scène, il faut le faire dans le main, après l'instanciation du moteur (le moteur instancie un SceneManager pour pouvoir gérer les scènes). Il suffit simplement d'instancier la classe comme d'habitude et de l'enregistrer auprès du SceneManager avec AddScene(nom, adresse).

```
int main(int argc, char** argv)
{
    Engine _engine = Engine(LAUNCH_MOD::GAME, 1920, 1080, "Space Explorer");

    SceneManager* _sceneManager = SceneManager::Instance();
    SceneMain _sceneMain = SceneMain();
    _sceneManager->AddScene("MAIN",&_sceneMain);

    Scene_Earth _earth = Scene_Earth();
    _sceneManager->AddScene("Earth",&_earth);

    Scene_Moon _moon = Scene_Moon();
    _sceneManager->AddScene("Moon",&_moon);

    Scene_Mars _mars = Scene_Mars();
    _sceneManager->AddScene("Mars",&_mars);

    _sceneManager->LoadScene("MAIN");
    _engine.Run();
    return 0;
}
```

FIGURE 3 – Exemple d'instanciation des scènes

2.4 Load

Pour load une scène, il suffit d'appeler LoadScene(nom) avec le nom de la scène à load (voir image ci-dessus).

3 Gameobjects

Les Gameobjects contiennent tous un pointeur sur leur parent (ou nullptr), une liste de pointeur sur les enfants, une liste de components ainsi qu'un transform. Le transform contient lui même la position, rotation et scale d'un gameobject.

3.1 Création

Pour créer un gameobject, il faut créer une classe héritant de GameObject.

```
class Ground_Player : public GameObject, public IMoving, public IRotating
{
    MeshComponent* mMeshComponent = nullptr;
    BoxCollider* mBoxCollider = nullptr;
    PhysicComponent* mPhysicComponent = nullptr;

    Camera* mPlayerCamera = nullptr;
```

FIGURE 4 – Exemple de création d'un GameObject Player

3.2 Instanciation

Pour instancier un GameObject, il ne faut pas faire de new. Nous devons utiliser la méthode Create<>() de l'ObjectManager. Cette méthode se charge de référencer ce GameObject auprès des Manager qui s'y intéresse (par exemple le LightManager si l'objet est une light).

```
ObjectManager* _objectManager = ObjectManager::Instance();
DirectionalLight* _sunLight = _objectManager->Create<DirectionalLight>();
_sunLight->SetColor(vec3(1));
_sunLight->SetDirection(vec3(1,-1,1));
```

FIGURE 5 – Exemple d'instanciation d'une light

3.3 Destruction

Pour détruire un GameObject, il ne faut pas faire de delete. Nous devons utiliser la méthode Destroy() de l'ObjectManager. Cette méthode se charge de marquer l'objet comme "A détruire". Il sera ensuite détruit à la prochaine frame avec ses components (cela évite de delete le gameobject directement alors qu'il pourrait être utiliser dans les calculs de collisions de la frame par exemple).

Nous avons donc les allocations et les destructions qui se font via une unique classe qui garde en mémoire les adresses des instances.

```
Asteroid* _randomAsteroid = mAsteroids[0];
_randomAsteroid->GetCollider()->SetOnCollisionCallback([=](CollisionData _data)
{
    if(_randomAsteroid->IsMarkedForDestroy()) return;
    CreateTrophy(_randomAsteroid->GetWorldPosition());
    ObjectManager* _objectManager = ObjectManager::Instance();
    _objectManager->Destroy(_randomAsteroid);
});
```

FIGURE 6 – Exemple de destruction d'un astéroïde à la collision

4 Components

4.1 Création

Pour créer un component, il faut créer une classe héritant de Component.

```
class Collider : public Component, public IRenderable
{
protected:
    bool mCanBeRendered = false;
    bool mIsTrigger = false;
    map<Collider*, CollisionData> mLastFrameColliderInContact = map<Collider*, CollisionData>();
    map<Collider*, CollisionData> mColliderInContact = map<Collider*, CollisionData>();

    function<void(CollisionData)> mCollisionEnterCallback = nullptr;
    function<void(CollisionData)> mCollisionStayCallback = nullptr;
    function<void(CollisionData)> mCollisionExitCallback = nullptr;

    function<void(CollisionData)> mTriggerEnterCallback = nullptr;
    function<void(CollisionData)> mTriggerStayCallback = nullptr;
    function<void(CollisionData)> mTriggerExitCallback = nullptr;
```

FIGURE 7 – Exemple d'un Collider

4.2 Instanciation

Pour instancier un Component, il ne faut pas faire de new. Nous devons utiliser la méthode AddComponent<>() d'un GameObject. Cette méthode se charge de référencer ce Component auprès des Manager qui s'y intéresse (par exemple le Render si l'objet est un MeshComponent (implémente IRenderable)).

```
Ground_Player::Ground_Player()
{
    mMeshComponent = AddComponent<MeshComponent>(vec3(0), vec3(0), vec3(.1));
    mBoxCollider = AddComponent<BoxCollider>();
    mBoxCollider->SetSize(vec3(10,30,10));

    mPhysicComponent = AddComponent<PhysicComponent>();
```

FIGURE 8 – Exemple d'instanciation de components

4.3 Récupérer un Component

Nous pouvons utiliser la méthode GetComponent<>() d'un GameObject. Cette méthode renvoie le premier component de type T ou nullptr si le Gamobject n'a aucun component de type T.

```cpp
void PhysicComponent::PostConstructor()
{
    Component::PostConstructor();
    if(!mOwner) return;
    mCollider = mOwner->GetComponent<Collider>();
    mReadyToCollide = false;
}
```

FIGURE 9 – Exemple de récupération du Collider du Owner par le PhysicComponent

4.4 Détruire des Components

Pour détruire un Component, il ne faut pas faire de delete. Nous devons utiliser la méthode DeleteComponent<>() d'un GameObject. Cette méthode permet de détruire les components de type T d'un gameobject. Les components seront automatiquement détruit à la destruction du gameobject. Nous n'avons pas utilisé cette méthode donc voici le code.

```cpp
template<typename T>
void GameObject::DeleteComponent()
{
    for(Component* _component : mComponents)
    {
        T* _castedComp = dynamic_cast<T*>(_component);
        if(!_castedComp) continue;
        mComponents.erase(remove(mComponents.begin(), mComponents.end(), _component), mComponents.end());
        delete _component;
    }
}
```

FIGURE 10 – Delete Component

5 Meshs & Materials

5.1 Création

Pour créer un Mesh/Material, il faut créer une classe héritant de Mesh/Material.

```cpp
class Plane : public Mesh
{
protected:
    int mNbVertexWidth = 0;
    int mNbVertexLength = 0;
```

FIGURE 11 – Exemple d'un Mesh

5.2 Instanciation

Pour instancier un mesh ou un material, il ne faut pas faire de new. Nous devons utiliser la méthode CreateMesh<>()/CreateMaterial<>() d'un MeshComponent.

```
Planet::Planet()
{
    mMeshComponent = AddComponent<MeshComponent>();
    mMeshComponent->CreateMesh<Sphere>();
    mMeshComponent->CreateMaterial<Material>(PHONG_VERTEX, PHONG_FRAG);
```

FIGURE 12 – Exemple d'instanciation d'un Mesh/Material

Les meshs/materials seront détruit si on les remplace ou à la destruction du MeshComponent.

5.3 Chargement d'un modèle 3D

Nous pouvons utiliser Assimp afin de charger un modèle 3D ainsi que ses materials. Un modèle Assimp est une hiérarchie de meshs avec, pour chacun de ses meshs, un material avec des textures associé. Pour charger un modèle et ses materials, il suffit d'utiliser CreateMesh<Mesh>() d'un MeshComponent().

```
mMeshComponent = AddComponent<MeshComponent>(vec3(0,0,0), vec3(0), vec3(1));
mMeshComponent->CreateMesh<Mesh>("3DModels/Spaceship/scene.gltf");
```

FIGURE 13 – Chargement d'un modèle (Meshs et Materials) d'un vaisseau

Les textures doivent se trouver dans un dossier "Textures".

FIGURE 14 – Textures du vaisseau

Parfois Assimp fournit un path de textures qui ne correspond à rien, il faut donc surcharger les materials en créant un nouveau material.

```
Trophy::Trophy()
{
    mMeshComponent = AddComponent<MeshComponent>(vec3(0), vec3(0), vec3(7));
    mMeshComponent->CreateMesh<Mesh>("3DModels/Trophy/Trophy.glb", false);
    mMeshComponent->CreateMaterial<Material>();
    Material* _material = mMeshComponent->GetMaterial(0);
    _material->SetTexture(TEXTURE_SLOT::ALBEDO, "3DModels/Trophy/Textures/Albedo.png");
```

FIGURE 15 – Surcharge du material d'index 0 du Gameobject Trophy

6 Inputs

6.1 Axis

Pour bind un axis, nous pouvons appeler la méthode BindAxis() de l'InputManager. Ce BindAxis() prend des couples d'inputs (le premier est la valeur positive, par exemple Z pour avancer, et le second est négatif, S pour reculer). La méthode prend aussi une instance d'une classe qui hérite de Object ainsi que la méthode de callback (castée) à appliquer sur l'instance avec un float qui sera entre -1 et 1 (-1,0,1 si on a clavier mais des valeurs décimales de -1 à 1 si on a un joystick).

```
InputManager* _inputManager = InputManager::Instance();
_inputManager->BindAxis({{GLFW_KEY_W,GLFW_KEY_S}}, mCharacter, (void* (Object::*)(float))&Ground_Player::MoveForwardBackward);
_inputManager->BindAxis({{GLFW_KEY_D,GLFW_KEY_A}}, mCharacter, (void* (Object::*)(float))&Ground_Player::MoveLateral);
_inputManager->BindKey(GLFW_KEY_SPACE, ACTION_TYPE::PRESS, mCharacter, (void* (Object::*)(bool))&Ground_Player::Jump);
```

FIGURE 16 – Bind d'axis et key

6.2 Keys

Pour bind une key, nous pouvons appeler la méthode BindKey() de l'InputManager. Ce BindKey() prend un input ainsi qu'un mode de déclenchement d'input (PRESS si on veut délcencher le callback à la frame où la touche est pressée, HOLD à toutes les frames où la touche est pressée et RELEASE à la frame où la touche est relâchée. Le reste de la méthode est comme BindAxis() à part que les méthodes de callback prennent des booléens (et non des floats) pour savoir si la touche est PRESS/HOLD/RELEASE selon le mode de bind (voir plus haut exemple).

6.3 Mouse

Pour utiliser la souris comme input, nous pouvons utiliser MOUSE_X (pour le déplacement horizontal de la souris) et MOUSE_Y (pour le déplacement vertical de la souris). Il faut bind deux fois le même input pour une paire (le mouvement est géré par le MouseManager est défini si le float est négatif ou positif).

```
InputManager::Instance()->BindAxis({{MOUSE_Y, MOUSE_Y}}, mCharacter, (void* (Object::*)(float))&Ground_Player::RotateLocalAxisX);
InputManager::Instance()->BindAxis({{MOUSE_X, MOUSE_X}}, mCharacter, (void* (Object::*)(float))&Ground_Player::RotateLocalAxisY);
```

FIGURE 17 – Bind d'axis en utilisant le mouvement de la souris

7 Camera

7.1 Création

Pour créer une camera, il faut créer une classe héritant de Camera.

```
class EditorCamera final : public Camera, public IMoving, public IRotating, public Singleton<EditorCamera>
{
public:
    EditorCamera();
    EditorCamera(const vec3& _position, const vec3& _rotation);
```

FIGURE 18 – Camera de l'engine

7.2 Activer/Désactiver

Pour render une scène à travers une caméra, il faut l'utiliser comme ViewportCamera. Par défaut, la caméra du viewport est une EditorCamera crée par le moteur. Il doit toujours y avoir une caméra d'active pour render une scène. Lorsqu'une caméra est activée, la caméra actuelle est désactivée.

```
class Engine final : public Singleton<Engine>
{
    LAUNCH_MOD mLaunchMod;

    float mDeltaTime = 0.0f;    // time between current frame and last frame
    float mLastFrame = 0.0f;

    Window* mMainWindow = nullptr;
    Camera* mActiveCamera = nullptr;

    VAO mMainVAO;

    EditorCamera* mEditorCamera = nullptr;
    Skybox* mSkybox = nullptr;
    InputManager mInputManager;
    MouseManager mMouseManager;
    ObjectManager mObjectManager;
    Renderer mRenderer;
    ShaderManager mShaderManager;
    SceneManager mSceneManager;
    LightManager mLightManager;
    PhysicManager mPhysicManager;
    GameManager mGameManager;
```

FIGURE 19 – Caméra par défaut

```
mPlayerCamera = CreatePlayerCamera();
Engine::Instance()->SetViewportCamera(mPlayerCamera);
```

FIGURE 20 – Activer une caméra

8 Collisions

8.1 Callbacks

Il est possible de bind des fonctions qui se déclencheront lors de collisions. Un collider peut être un trigger (aucune réponse physique à une collision) ou un collider brut (réponse physique). Nous disposons de OnTriggerEnter, OnTriggerStay et OnTriggerExit qui sont des std : :function qui se déclenchent, si le collider est un trigger, à la frame où un élément rentre dans le trigger, à chaque frame où un élémént est dans le trigger et à la frame où un élément sort du trigger. Pour le collider brut, nous possédons OnCollide qui se déclenche lorsque le collider rentre en collision avec un autre collider brut.

```
_trophy->GetCollider()->SetOnTriggerEnterCallback([=](CollisionData _data)
{
    mCollectibleAvailable = false;
    _objectManager->Destroy(_trophy);
});
```

FIGURE 21 – Exemple de callback sur le OnTriggerEnter d'un collider

8.2 Filtre de collisions

Il est possible de filtrer les collisions selon le type de collision. Nous pouvons labelliser un collider et ajouter plusieurs labels à ses labels à ignorer. Deux labels qui s'ignorent ne rentrerons pas collision (ni collision ni trigger) et ne déclencherons pas les callbacks.

```
mPhysicComponent->SetCollisionType(COLLISION_TYPE::TROPHY);
mPhysicComponent->AddIgnoredCollisionType({COLLISION_TYPE::PLANET, COLLISION_TYPE::ASTEROID, COLLISION_TYPE::PROJECTILE});
```

FIGURE 22 – Labellisation de la classe Trophy et ajout des labels à ignorer

9 Skybox

La skybox est automatiquement crée par l'engine. Cette classe est "final" donc nous ne pouvons pas créer de classe qui en hérite. De plus nous ne pouvons pas instancier cette classe, l'engine le fait lors de son constructeur et la détruit lors de son destructeur.

Nous pouvons néanmoins changer l'apparence de la skybox en utilisant ChangeSkybox() qui prend le path de la skybox et l'extension. Les faces de la skybox doivent être correctement nommées.

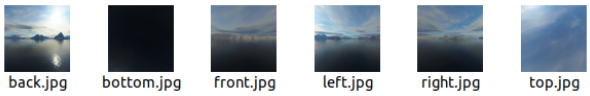

FIGURE 23 – Nomenclature

```
void Scene_Moon::LoadScene()
{
    Skybox* _skybox = Skybox::Instance();
    _skybox->ChangeSkybox("Textures/Skybox/Moon/", ".png");
```

FIGURE 24 – Changer la skybox

10 Terrains

Les terrains possèdent des matériaux spéciaux (pas de metalness, roughness,...) mais un maximum de 16 couches. Nous pouvons aussi utiliser ApplyHeightmap(path, heightMax, shift) afin d'appliquer une heightmap à notre terrain. Nous avons aussi ChangeResolution() et d'autres méthodes (sur le Landscape Material) comme SetTiling() pour paramétrer le tiling des textures, AddTransition() pour modifier les transitions entre textures, ...

```
LandscapeMaterial* _landscapeMaterial= _landscape->GetMaterial();
_landscapeMaterial->AddLayer(0, GRASS_TEXTURE);
_landscapeMaterial->AddLayer(1, ROCK_TEXTURE);
_landscapeMaterial->AddLayer(2, SNOWROCKS_TEXTURE);
```

FIGURE 25 – Changer les layers d'un terrain