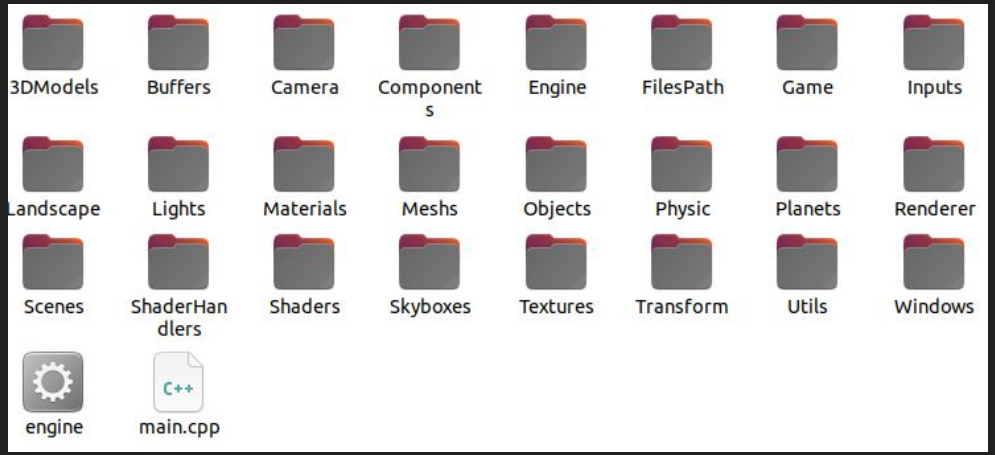
Moteur de Jeux

Ingo DIAB
Yahnis SAINT-VAL
Arthur VILLARROYA-PALAU

Sommaire

Aspect général

- Environ 60 classes (inputs, scenes, gameobjects, materials, …)



Aspect général

- Environ 60 classes (inputs, scenes, gameobjects, materials, …)
- Managers qui héritent de Singleton => Il ne peut exister qu'une seule instance de cette classe

```
class SceneManager final : public Singleton<SceneManager>
```

Aspect général

- Environ 60 classes (inputs, scenes, gameobjects, materials, …)
- Managers qui héritent de Singleton => Il ne peut exister qu'une seule instance de cette classe

```
class SceneManager final : public Singleton<SceneManager>
```

- Classes abstraites/interfaces (IRenderable, …) qui permettent de regrouper certaines entités ayant les mêmes caractéristiques

```
public:
    virtual void Render(Camera* _renderingCamera) = 0;
    virtual void CleanRessources() = 0;
```

Aspect général

- Environ 60 classes (inputs, scenes, gameobjects, materials, …)
- Managers qui héritent de Singleton => Il ne peut exister qu'une seule instance de cette classe

```
class SceneManager final : public Singleton<SceneManager>
```

- Classes abstraites/interfaces (IRenderable, …) qui permettent de regrouper certaines entités ayant les mêmes caractéristiques

```
public:
    virtual void Render(Camera* _renderingCamera) = 0;
    virtual void CleanRessources() = 0;
```

- Le moteur utilise un repère left-handed

Inputs

- 2 Types d’inputs : Key et Axis

Inputs

- 2 Types d’inputs : Key et Axis
- Axis : combinaison de deux boutons, s’ils sont pressés le premier renverra 1 et le second -1 (par exemple si on bind l’axis D/Q, D renverra 1 et Q renverra -1).
- On utilise les Axis surtout pour les déplacements

Inputs

- 2 Types d’inputs : Key et Axis
- Axis : combinaison de deux boutons, s’ils sont pressés le premier renverra 1 et le second -1 (par exemple si on bind l’axis D/Q, D renverra 1 et Q renverra -1).
- On utilise les Axis surtout pour les déplacements
- Key : 3 modes de pression d’un bouton : PRESS (la frame où le bouton est appuyé), HOLD (les frames entre le moment où le bouton est appuyé et où il est relâché) et RELEASE (la frame où le bouton est relâché)
- On l’utilise pour des actions ponctuelles (tirer, interagir avec le décor,...)

Inputs

- Bind des actions à des Keys/Axis

Inputs

- Bind des actions à des Keys/Axis
- Méthodes de callback qui seront déclenchées quand il le faut

Inputs

- Bind des actions à des Keys/Axis
- Méthodes de callback qui seront déclenchées quand il le faut
- Renvoient “void” et prennent un “bool” (si on le bind à une Keys) ou un “float” (si bind à un Axis)

Inputs

- Bind des actions à des Keys/Axis
- Méthodes de callback qui seront déclenchées quand il le faut
- Renvoient “void” et prennent un “bool” (si on le bind à une Keys) ou un “float” (si bind à un Axis)
- A chaque début d’update du moteur, on vérifie si les keys/axis sont appuyés. Si c’est le cas, on vérifie leur état (pour les keys) et on déclenche la méthode de callback.

Inputs

- Bind des actions à des Keys/Axis
- Méthodes de callback qui seront déclenchées quand il le faut
- Renvoient “void” et prennent un “bool” (si on le bind à une Keys) ou un “float” (si bind à un Axis)
- A chaque début d’update du moteur, on vérifie si les keys/axis sont appuyés. Si c’est le cas, on vérifie leur état (pour les keys) et on déclenche la méthode de callback.
- Exemple de bind des contrôles du joueur :

```
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);
```

GameObjects

- Objets dont héritent tous les objets de la scène (character, décor, …)

GameObjects

- Objets dont héritent tous les objets de la scène (character, décor, …)
- Contient une Durabilité, un Transform, un Parent (nullptr s’il n’en a pas), une liste d’Enfants et une liste de Components.

GameObjects

- Objets dont héritent tous les objets de la scène (character, décor, …)
- Contient une Durabilité, un Transform, un Parent (nullptr s'il n'en a pas), une liste d'Enfants et une liste de Components.
- Durabilité : SCENE ou PERSISTENT
  - SCENE = unload en même temps que la scène (ex: asteroids dans l'espace)
  - PERSISTENT = unload quand le moteur s'arrête (ex: caméra de l'Engine)

GameObjects

- Objets dont héritent tous les objets de la scène (character, décor, …)
- Contient une Durabilité, un Transform, un Parent (nullptr s'il n'en a pas), une liste d'Enfants et une liste de Components.
- Durabilité : SCENE ou PERSISTENT
  - SCENE = unload en même temps que la scène (ex: asteroids dans l'espace)
  - PERSISTENT = unload quand le moteur s'arrête (ex: caméra de l'Engine)
- Transform : Gère les transformations de l'objet

GameObjects

- Objets dont héritent tous les objets de la scène (character, décor, …)
- Contient une Durabilité, un Transform, un Parent (nullptr s'il n'en a pas), une liste d'Enfants et une liste de Components.
- Durabilité : SCENE ou PERSISTENT
  - SCENE = unload en même temps que la scène (ex: asteroids dans l'espace)
  - PERSISTENT = unload quand le moteur s'arrête (ex: caméra de l'Engine)
- Transform : Gère les transformations de l'objet
- Parent/Enfant : Transformations relatives à la hiérarchie

GameObjects

- Objets dont héritent tous les objets de la scène (character, décor, …)
- Contient une Durabilité, un Transform, un Parent (nullptr s'il n'en a pas), une liste d'Enfants et une liste de Components.
- Durabilité : SCENE ou PERSISTENT
  - SCENE = unload en même temps que la scène (ex: asteroids dans l'espace)
  - PERSISTENT = unload quand le moteur s'arrête (ex: caméra de l'Engine)
- Transform : Gère les transformations de l'objet
- Parent/Enfant : Transformations relatives à la hiérarchie
- Components : Mesh, Collider, …

GameObjects

- ObjectManager : Singleton qui possède une référence sur chaque objet créé

GameObjects

- ObjectManager : Singleton qui possède une référence sur chaque objet créé
- Créer un gameObject : Appelle à la méthode “Create<T>” de ObjectManager

GameObjects

- ObjectManager : Singleton qui possède une référence sur chaque objet créé
- Créer un gameObject : Appelle à la méthode “Create<T>” de ObjectManager
  - Utiliser des dynamic_cast pour “ranger” les objets où ils doivent être

```
IRenderable* _gameobjectRenderable = dynamic_cast<IRenderable*>(_gameobject);
if(_gameobjectRenderable) Renderer::Instance()->AddRenderable(_gameobjectRenderable);
```

GameObjects

- ObjectManager : Singleton qui possède une référence sur chaque objet créé
- Créer un gameObject : Appelle à la méthode “Create<T>” de ObjectManager
  - Utiliser des dynamic_cast pour “ranger” les objets où ils doivent être

```
IRenderable* _gameobjectRenderable = dynamic_cast<IRenderable*>(_gameobject);
if(_gameobjectRenderable) Renderer::Instance()->AddRenderable(_gameobjectRenderable);
```

  - ObjectManager gère la mémoire (il est seul à faire un new et à faire un delete)

GameObjects

- ObjectManager : Singleton qui possède une référence sur chaque objet créé
- Créer un gameObject : Appelle à la méthode “Create<T>” de ObjectManager
  - Utiliser des dynamic_cast pour “ranger” les objets où ils doivent être

```
IRenderable* _gameobjectRenderable = dynamic_cast<IRenderable*>(_gameobject);
if(_gameobjectRenderable) Renderer::Instance()->AddRenderable(_gameobjectRenderable);
```

  - ObjectManager gère la mémoire (il est seul à faire un new et à faire un delete)
- Exemple de création d’un GameObject

```
ObjectManager* _objectManager = ObjectManager::Instance();
Planet* _earth = _objectManager->Create<Planet>(vec3(0), vec3(0,0,23.44f), vec3(.1f), _parent);
```

GameObjects

- Détruire un gameObject : Appelle à la méthode “Destroy” de ObjectManager
- Le GameObject est marqué comme “à détruire” et placé dans une liste prévue à cet effet.
- En début de frame suivante, tous les objets marqués comme “à détruire” sont supprimé (avec leur components)

Components

- 3 components : MeshComponent, Collider et PhysicComponent

Components

- 3 components : MeshComponent, Collider et PhysicComponent
  - MeshComponent : contient plusieurs meshs et maximum 1 material par mesh

Components

- 3 components : MeshComponent, Collider et PhysicComponent
  - MeshComponent : contient plusieurs meshs et maximum 1 material par mesh
  - Collider : Box ou Sphere pour les collisions

Components

- 3 components : MeshComponent, Collider et PhysicComponent
  - MeshComponent : contient plusieurs meshs et maximum 1 material par mesh
  - Collider : Box ou Sphere pour les collisions
  - PhysicComponent : Reconnu par le moteur physique (permet d’appliquer une vélocité, la gravité, force de frottement, clip au terrain et collisions mais nécessite un collider).

Components

- 3 components : MeshComponent, Collider et PhysicComponent
  - MeshComponent : contient plusieurs meshs et maximum 1 material par mesh
  - Collider : Box ou Sphere pour les collisions
  - PhysicComponent : Reconnu par le moteur physique (permet d'appliquer une vélocité, la gravité, force de frottement, clip au terrain et collisions mais nécessite un collider).
- Chaque component possède une référence sur le GameObject qui le possède (Owner)

Components

- Ajouter un component : appelle de “AddComponent<T>” sur le gameObject
  - Utilisation de dynamic_cast pour les ajouter aux managers (comme pour les gameObjects)

Components

- Ajouter un component : appelle de “AddComponent<T>” sur le gameObject
  - Utilisation de dynamic_cast pour les ajouter aux managers (comme pour les gameObjects)
- Modifier un component : appelle de “GetComponent<T>” sur le gameObject
  - Retourne le premier component de type T de l’objet (ou nullptr s’il n’y en a pas)

Components

- Ajouter un component : appelle de “AddComponent<T>” sur le gameObject
  - Utilisation de dynamic_cast pour les ajouter aux managers (comme pour les gameObjects)
- Modifier un component : appelle de “GetComponent<T>” sur le gameObject
  - Retourne le premier component de type T de l’objet (ou nullptr s’il n’y en a pas)
- Supprimer un component : appelle de “DeleteComponent<T>” sur le gameObject

Mesh Component

- Implémente IRenderable (méthodes pour render, méthode pour clean les ressources) et hérite de Component

Mesh Component

- Implémente IRenderable (méthodes pour render, méthode pour clean les ressources) et hérite de Component
- Création de mesh et de materials

Mesh Component

- Implémente IRenderable (méthodes pour render, méthode pour clean les ressources) et hérite de Component
- Création de mesh et de materials
- Render le component : s’il le component peut être render, on parcours tous les mesh du components et on les render en utilisant le material associé dans la liste des matérial du component.

Mesh Component

- Implémente IRenderable (méthodes pour render, méthode pour clean les ressources) et hérite de Component
- Création de mesh et de materials
- Render le component : s’il le component peut être render, on parcours tous les mesh du components et on les render en utilisant le material associé dans la liste des matérial du component.
- Le Renderer (manager singleton) s’occupe de render tous les objets dont il a la référence.

Mesh Component

- Création d’un mesh : Méthode “CreateMesh<T>” du MeshComponent
  - Permet de créer des primitives héritant de Mesh (Sphère, Cube, …)
  - Permet de load des meshs .gltf/.glb avec Assimp et les materials qui vont avec (Chargement récursif des meshs contenu dans un .gltf/.glb)

Mesh Component

- Création d’un mesh : Méthode “CreateMesh<T>” du MeshComponent
  - Permet de créer des primitives héritant de Mesh (Sphère, Cube, …)
  - Permet de load des meshs .gltf/.glb avec Assimp et les materials qui vont avec (Chargement récursif des meshs contenu dans un .gltf/.glb)
- Création d’un material : Méthode “CreateMaterial<T>” du MeshComponent
  - Permet de créer un material paramétrable (shader, textures, couleurs, coefficients, …)
  - Peu utile si on load les meshs via Assimp

Mesh

- Plusieurs VBO (positions, normales, uvs, indices, …)
- Méthode “RefreshVBO” qui permet de mettre à jour un/des VBO à partir des arrays correspondants

Material

- BaseMaterial contient une référence sur un ShaderHandler

Material

- BaseMaterial contient une référence sur un ShaderHandler
- 2 types de matérials héritent de BaseMaterial : LandscapeMaterial et Material

Material

- BaseMaterial contient une référence sur un ShaderHandler
- 2 types de matérials héritent de BaseMaterial : LandscapeMaterial et Material
- LandscapeMaterial : utilisé pour les terrains
    - Contient une heightmap, une height max, un shift (pour baisser le terrain), une liste de layers (terre, roche, neige, …), les transitions qui correspondent aux layers, un threshold pour des transitions plus ou moins étendues et un coefficient de tiling.

Material

- BaseMaterial contient une référence sur un ShaderHandler
- 2 types de matérials héritent de BaseMaterial : LandscapeMaterial et Material
- LandscapeMaterial : utilisé pour les terrains
  - Contient une heightmap, une height max, un shift (pour baisser le terrain), une liste de layers (terre, roche, neige, …), les transitions qui correspondent aux layers, un threshold pour des transitions plus ou moins étendues et un coefficient de tiling.
- Material : utilisé pour le reste
  - Contient 3 couleurs (ambiant, diffuse, specular) , 7 emplacement de textures (albedo, normalmap,...), 7 coefficients (metalness, roughness, …).

Material

- BaseMaterial contient une référence sur un ShaderHandler
- 2 types de matérials héritent de BaseMaterial : LandscapeMaterial et Material
- LandscapeMaterial : utilisé pour les terrains
  - Contient une heightmap, une height max, un shift (pour baisser le terrain), une liste de layers (terre, roche, neige, …), les transitions qui correspondent aux layers, un threshold pour des transitions plus ou moins étendues et un coefficient de tiling.
- Material : utilisé pour le reste
  - Contient 3 couleurs (ambiant, diffuse, specular) , 7 emplacement de textures (albedo, normalmap,...), 7 coefficients (metalness, roughness, …).
- Les ShaderHandler récupèrent les uniforms des shaders et les Materials envoient les données au Shader via le ShaderHandler.

Material

- BaseMaterial contient une référence sur un ShaderHandler
- 2 types de matérials héritent de BaseMaterial : LandscapeMaterial et Material
- LandscapeMaterial : utilisé pour les terrains
  - Contient une heightmap, une height max, un shift (pour baisser le terrain), une liste de layers (terre, roche, neige, …), les transitions qui correspondent aux layers, un threshold pour des transitions plus ou moins étendues et un coefficient de tiling.
- Material : utilisé pour le reste
  - Contient 3 couleurs (ambiant, diffuse, specular) , 7 emplacement de textures (albedo, normalmap,...), 7 coefficients (metalness, roughness, …).
- Les ShaderHandler récupèrent les uniforms des shaders et les Materials envoient les données au Shader via le ShaderHandler.
- En plus des données du Material, on envoie aussi des données de la scène (les lights du LightManager, la caméra active, la skybox, …)

BoxCollider

- Boite englobante OBB

BoxCollider

- Boite englobante OBB
- Projection sur les axes (boîte englobante AABB)

BoxCollider

- Boite englobante OBB
- Projection sur les axes (boîte englobante AABB)
- 2 vecteur3D : un pour le centre du collider et un pour la taille sur chacun des axes

BoxCollider

- Boite englobante OBB
- Projection sur les axes (boîte englobante AABB)
- 2 vecteur3D : un pour le centre du collider et un pour la taille sur chacun des axes
- Booléen pour savoir si le collider est un trigger ou un collider

BoxCollider

- Boite englobante OBB
- Projection sur les axes (boîte englobante AABB)
- 2 vecteur3D : un pour le centre du collider et un pour la taille sur chacun des axes
- Booléen pour savoir si le collider est un trigger ou un collider
- Trigger : 3 méthodes de callback (std::function)
  - TriggerEnter : Déclenchée la frame où un objet rentre dans ce collider
  - TriggerStay : Déclenchée chaque frame où un objet est dans le collider
  - TriggerExit : Déclenchée la frame où un objet sort du collider

BoxCollider

- Boite englobante OBB
- Projection sur les axes (boîte englobante AABB)
- 2 vecteur3D : un pour le centre du collider et un pour la taille sur chacun des axes
- Booléen pour savoir si le collider est un trigger ou un collider
- Trigger : 3 méthodes de callback (std::function)
  - TriggerEnter : Déclenchée la frame où un objet rentre dans ce collider
  - TriggerStay : Déclenchée chaque frame où un objet est dans le collider
  - TriggerExit : Déclenchée la frame où un objet sort du collider
- Collider : 1 méthode de callback (std::function)
  - Collision : Déclenchée lorsque deux objets entrent en collision

PhysicComponent

- Nécessite un collider (méthode “PostConstructor” qui sera appelée après le constructor du component pour prendre la référence du collider de l’owner)

PhysicComponent

- Nécessite un collider (méthode “PostConstructor” qui sera appelée après le constructor du component pour prendre la référence du collider de l’owner)
- Plusieurs paramètres : forces de frictions, masse, bounciness, statique ou pas, …

PhysicComponent

- Nécessite un collider (méthode “PostConstructor” qui sera appelée après le constructor du component pour prendre la référence du collider de l’owner)
- Plusieurs paramètres : forces de frictions, masse, bounciness, statique ou pas, …
- Un vecteur3D vélocité et un float vélocitéMax

PhysicComponent

- Nécessite un collider (méthode “PostConstructor” qui sera appelée après le constructor du component pour prendre la référence du collider de l’owner)
- Plusieurs paramètres : forces de frictions, masse, bounciness, statique ou pas, un booléen pour savoir si l’objet est affecté par la gravité…
- Un vecteur3D vélocité et un float vélocitéMax
- Ce component permet d’appliquer une vélocité, de clipper un objet sur le terrain et prendre en compte les collisions avec le collider

Vélocité

- Calcul de l’accélération (avec ou sans l’accélération gravitationnelle)

Vélocité

- Calcul de l’accélération (avec ou sans l’accélération gravitationnelle)
- Calcul de la vélocité avec l’accélération

Vélocité

- Calcul de l’accélération (avec ou sans l’accélération gravitationnelle)
- Calcul de la vélocité avec l’accélération
- On va réduire l’accélération et la vélocité avec des forces de frottements

Vélocité

- Calcul de l’accélération (avec ou sans l’accélération gravitationnelle)
- Calcul de la vélocité avec l’accélération
- On va réduire l’accélération et la vélocité avec des forces de frottements
- En dessous d’un threshold, les vecteurs sont remis à 0

Vélocité

- Calcul de l’accélération (avec ou sans l’accélération gravitationnelle)
- Calcul de la vélocité avec l’accélération
- On va réduire l’accélération et la vélocité avec des forces de frottements
- En dessous d’un threshold, les vecteurs sont remis à 0
- On clamp la vélocité à la taille vélocitéMax et on ajoute la vélocité à la position de l’objet

Clip sur le terrain

- Projection de la position du Owner sur le terrain
- Calculs pour trouver la height de la heightmap correspondant à la projection
- Placer le Owner à la position du terrain + height + offset

Collisions

- Détection des Collisions AABB (Broad Phase)
    - On crée des CollisionData contenant les PhysicComponent qui se collide, la normale de la collision, …

Collisions

- Détection des Collisions AABB (Broad Phase)
    - On crée des CollisionData contenant les PhysicComponent qui se collide, la normale de la collision, …
- Résolutions des Collisions
    - On parcours toutes les CollisionData crée
    - Seulement si les deux colliders ne sont pas triggers
    - Déclenchement des callbacks selon les cas

Collisions

- Détection des Collisions AABB (Broad Phase)
  - On crée des CollisionData contenant les PhysicComponent qui se collide, la normale de la collision, …
- Résolutions des Collisions
  - On parcours toutes les CollisionData crée
  - Seulement si les deux colliders ne sont pas triggers
  - Déclenchement des callbacks selon les cas
- Système de Type de Collision (Planètes, Projectile, …)
  - Permet de faire filtrer les collisions (par exemple les planètes ignorent les collisions avec les objets de type projectiles,...)

Camera

- Hérite de GameObject

Camera

- Hérite de GameObject
- Une unique caméra active dans le moteur (Camera Viewport)

Camera

- Hérite de GameObject
- Une unique caméra active dans le moteur (Camera Viewport)
- On peut facilement changer la Camera Viewport

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

Camera

- Hérite de GameObject
- Une unique caméra active dans le moteur (Camera Viewport)
- On peut facilement changer la Camera Viewport

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

- Tous les objets qui seront render, seront render avec la View/Projection Matrix de la Camera Viewport

Scene

- SceneManager qui gère toutes les scènes

Scene

- SceneManager qui gère toutes les scènes
- Permet de load une scène grâce à son nom (ce qui unload la scène actuelle)

```
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);

_sceneManager->LoadScene("MAIN");
```

Scene

- SceneManager qui gère toutes les scènes
- Permet de load une scène grâce à son nom (ce qui unload la scène actuelle)

```
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);

_sceneManager->LoadScene("MAIN");
```

- Unload une scène supprime tous les gameObject ayant été créé avec une durabilité SCENE ainsi que tous les inputs dont l'instance appelante a été supprimé

Skybox

- Classe finale
- Méthode pour load une cubemap
- Draw
  - Model Matrix = Identity
  - Avant les autres objets
  - En désactivant la profondeur puis en la réactivant après le draw

Lights

- 2 types de lights : PointLight et Directional Light

Lights

- 2 types de lights : PointLight et Directional Light
- PointLight : lumière du soleil dans la scène Espace (placée à l’intérieur de la sphère)
  - Contient une couleur, une intensité et un bool enabled
  - Dans les shaders, l’illumination est atténuée par la distance carré entre le vertex et la lumière

Lights

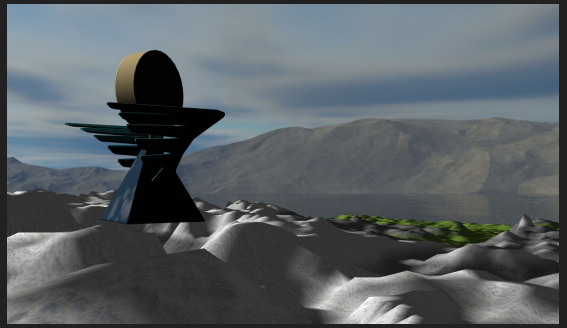
- 2 types de lights : PointLight et Directional Light
- PointLight : lumière du soleil dans la scène Espace (placée à l’intérieur de la sphère)
  - Contient une couleur, une intensité et un bool enabled
  - Dans les shaders, l’illumination est atténuée par la distance carré entre le vertex et la lumière
- DirectionalLight : lumière du soleil dans les autres scène
  - Contient une couleur, une direction et un bool enabled
  - Dans les shaders, l’illumination n’est pas atténuée

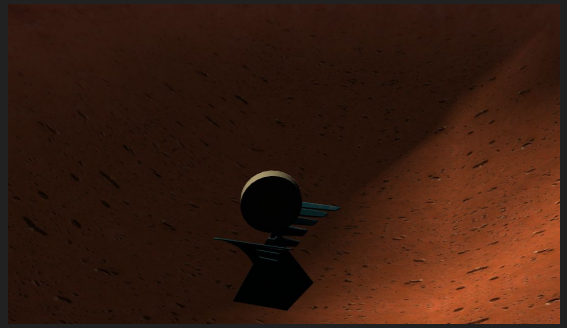
Lights

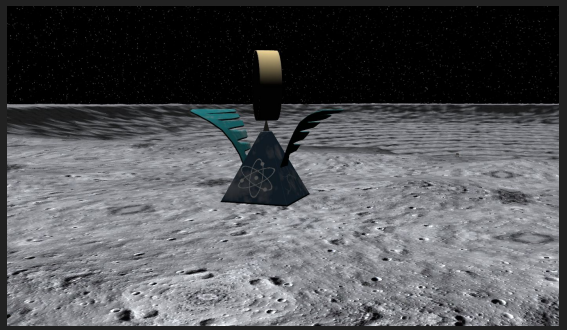
- 2 types de lights : PointLight et Directional Light
- PointLight : lumière du soleil dans la scène Espace (placée à l’intérieur de la sphère)
  - Contient une couleur, une intensité et un bool enabled
  - Dans les shaders, l’illumination est atténuée par la distance carré entre le vertex et la lumière
- DirectionalLight : lumière du soleil dans les autres scène
  - Contient une couleur, une direction et un bool enabled
  - Dans les shaders, l’illumination n’est pas atténuée
- LightManager pour gérer les lights

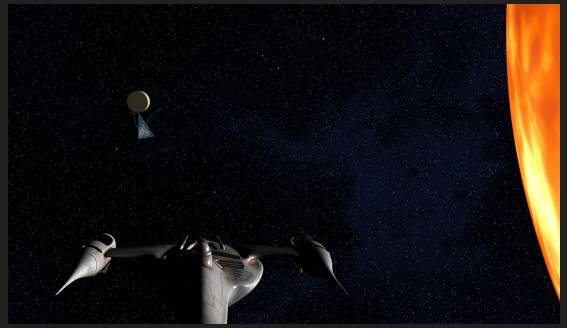
Jeu : Space Explorer

- But : Ramasser les 4 trophées cachés









Jeu : Space Explorer

- Deux parties : 1 partie Espace et 1 partie sur Terrain

Jeu : Space Explorer

- Deux parties : 1 partie Espace et 1 partie sur Terrain
- GameManager pour gérer le changement de scène
  - Le vaisseau du joueur possède un trigger
  - Les planètes possèdent un trigger
  - Lorsque les deux triggers collident, le joueur appuie sur une touche pour charger la scène correspondante à la planète.

Jeu : Space Explorer

- Partie Espace :
  - Des astéroïdes sont généré aléatoirements autour du soleil
  - L’un d’eux fait spawn le trophée à sa destruction
  - La destruction est une méthode de callback quand l'astéroïde entre collision avec un projectile

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

```
Trophy* SceneMain::CreateTrophy(const vec3& _position)
{
    if(!mCollectibleAvailable) return nullptr;
    ObjectManager* _objectManager = ObjectManager::Instance();
    Trophy* _trophy = _objectManager->Create<Trophy>(_position);
    _trophy->GetCollider()->SetOnTriggerEnterCallback([=](CollisionData _data)
    {
        mCollectibleAvailable = false;
        _objectManager->Destroy(_trophy);
    });

    return _trophy;
}
```

Jeu : Space Explorer

- Partie Terrain :
  - Le joueur peut courir en maintenant left shift
  - Accélération pour passer de la marche à la course
  - Décélération (plus grande que l’accélération) pour passer de la course à la marche
  - Le joueur peut sauter (le saut utilise la physique, lorsque la hauteur du joueur est inférieur à la height de la projection du joueur sur le terrain, on enlève la physique et on le reclip au terrain)
  - Après avoir ramassé le trophée caché, le joueur doit revenir au vaisseau et appuyer sur Q pour repartir dans l’espace.