ESIEE PARIS

Développement d'un moteur de jeu vidéo

Projet réalisé par :

CHOMICKI Clément • BARUZY Amaury • PARE Barbara

LEBLON Alexandre • MONTANI Maÿlis

ZHOU Sebastien • SALLET Aymeric

Projet encadré par :

Nabil MUSTAFA • Sylvain GLAIZE

une école de la CCI PARIS ILE-DE-FRANCE

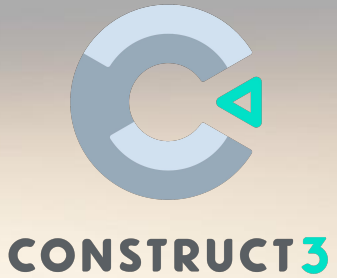
I. Etat de l’art

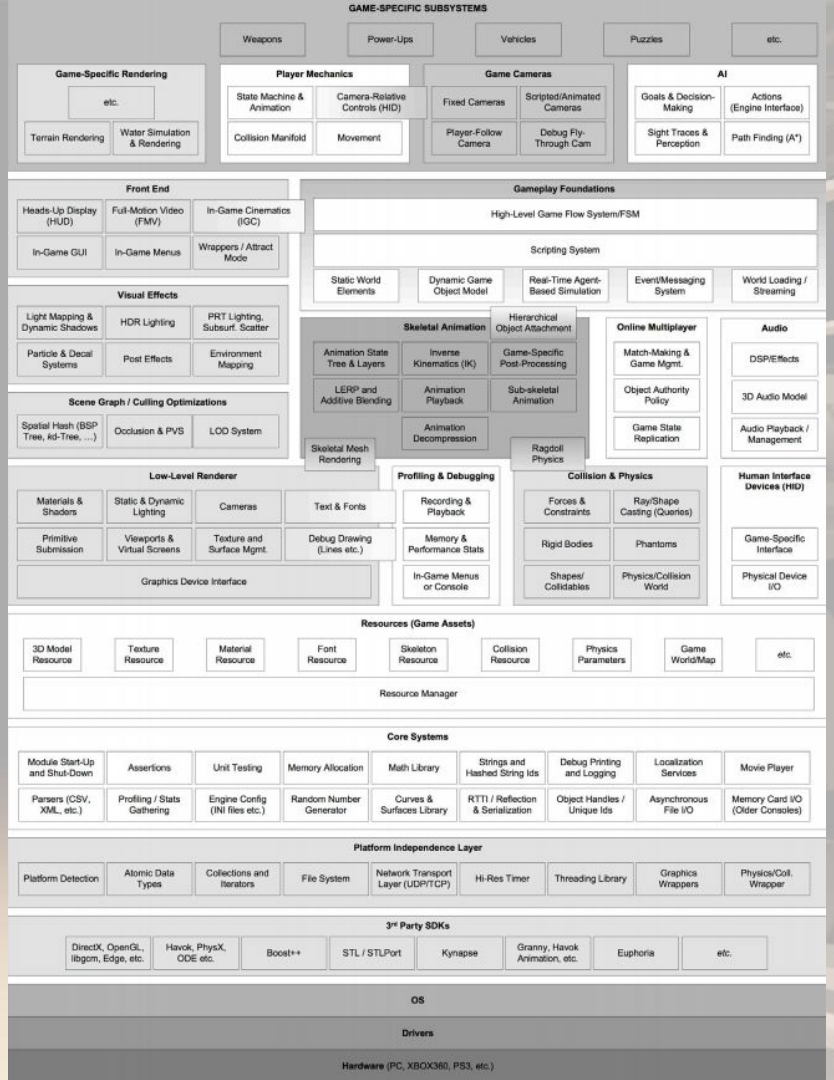
I.1. Qu’est ce qu’un moteur de jeu ?

I.2. Historique

I.1. Qu’est ce qu’un moteur de jeu ?

- Situé entre le jeu et la plateforme
- Partie bas niveau utilisée pour développer des jeux par dessus
- Multitude de moteurs de jeux vidéos en différents langages
- Constitué de différentes parties




GAME-SPECIFIC SUBSYSTEMS
Weapons
Power-Ups
Vehicles
Puzzles
etc.
Game-Specific Rendering
etc.
Terrain Rendering
Water Simulation & Rendering
Player Mechanics
State Machine & Animation
Camera-Relative Controls (HID)
Collision Manifold
Movement
Game Cameras
Fixed Cameras
Scripted/Animated Cameras
Player-Follow Camera
Debug Fly-Through Cam
AI
Goals & Decision-Making
Actions (Engine Interface)
Sight Traces & Perception
Path Finding (A*)
Front End
Heads-Up Display (HUD)
Full-Motion Video (FMV)
In-Game Cinematics (IGC)
In-Game GUI
In-Game Menus
Wrappers / Attract Mode
Visual Effects
Light Mapping & Dynamic Shadows
HDR Lighting
PRT Lighting, Subsurf. Scatter
Particle & Decal Systems
Post Effects
Environment Mapping
Scene Graph / Culling Optimizations
Spatial Hash (BSP Tree, kd-Tree, ...)
Occlusion & PVS
LOD System
Gameplay Foundations
High-Level Game Flow System/FSM
Scripting System
Static World Elements
Dynamic Game Object Model
Real-Time Agent-Based Simulation
Event/Messaging System
World Loading / Streaming
Skeletal Animation
Hierarchical Object Attachment
Animation State Tree & Layers
Inverse Kinematics (IK)
Game-Specific Post-Processing
LERP and Additive Blending
Animation Playback
Sub-skeletal Animation
Animation Decompression
Skeletal Mesh Rendering
Ragdoll Physics
Online Multiplayer
Match-Making & Game Mgmt.
Object Authority Policy
Game State Replication
Audio
DSP/Effects
3D Audio Model
Audio Playback / Management
Low-Level Renderer
Materials & Shaders
Static & Dynamic Lighting
Cameras
Text & Fonts
Primitive Submission
Viewports & Virtual Screens
Texture and Surface Mgmt.
Debug Drawing (Lines etc.)
Graphics Device Interface
Profiling & Debugging
Recording & Playback
Memory & Performance Stats
In-Game Menus or Console
Collision & Physics
Forces & Constraints
Ray/Shape Casting (Queries)
Rigid Bodies
Phantoms
Shapes/ Collidables
Physics/Collision World
Human Interface Devices (HID)
Game-Specific Interface
Physical Device I/O
Resources (Game Assets)
3D Model Resource
Texture Resource
Material Resource
Font Resource
Skeleton Resource
Collision Resource
Physics Parameters
Game World/Map
etc.
Resource Manager
Core Systems
Module Start-Up and Shut-Down
Assertions
Unit Testing
Memory Allocation
Math Library
Strings and Hashed String Ids
Debug Printing and Logging
Localization Services
Movie Player
Parsers (CSV, XML, etc.)
Profiling / Stats Gathering
Engine Config (INI files etc.)
Random Number Generator
Curves & Surfaces Library
RTTI / Reflection & Serialization
Object Handles / Unique Ids
Asynchronous File I/O
Memory Card I/O (Older Consoles)
Platform Independence Layer
Platform Detection
Atomic Data Types
Collections and Iterators
File System
Network Transport Layer (UDP/TCP)
Hi-Res Timer
Threading Library
Graphics Wrappers
Physics/Coll. Wrapper
3rd Party SDKs
DirectX, OpenGL, libgcm, Edge, etc.
Havok, PhysX, ODE etc.
Boost++
STL / STLPort
Kynapse
Granny, Havok Animation, etc.
Euphoria
etc.
OS
Drivers
Hardware (PC, XBOX360, PS3, etc.)

I.2. Historique

Plusieurs langages de programmation des moteurs de jeux vidéos

- Jeux en Assembleur

- Les premiers moteurs de jeu en C

ID Tech

id TECH

- Depuis plusieurs dizaines d'années : C++

Anvil next

- Depuis quelques années, on diversifie avec Python, C#, ou avec des langages plus récents comme Rust

blender GAME ENGINE

Amethyst
Game Engine
Data-driven game engine written in Rust

II. Décisions préalables

II.1. Pourquoi le langage Rust ?

II.2. Choix de la plateforme de développement

II.3. Mise en commun du travail & rapports de réunion

II.1. Pourquoi le langage Rust ?

- Langage “jeune”
- Meilleure gestion ownership, lifetime, typage
- Peu de moteurs de jeu vidéo en Rust → Innovant
- Clément connaît bien Rust, et c’était l’occasion apprendre un nouveau langage que nous ne verrions pas à l’école

piston

II.2. Choix de la plateforme de développement

- Essais laborieux sous Windows et Machines Virtuelles
- Plateforme choisie : Linux

- Environnement de développement : Emacs, Visual Studio Code et Gedit

II.3. Mise en commun du travail & rapports de réu

- Utilisation de GitHub → Travail de groupe et gestion des conflits de versions
- Rapports de réunion en Markdown sur Github
- Guides de prise en main des différents outils

III. Déroulement du projet

III.1. Répartition des tâches

III.2. ECS

III.3. Graphics

III.4. Physics

III.5. Graphical User Interface

III.6. Son

III.7. Jeu test

III.1. Répartition des tâches

- Clément Chef de projet & architecture

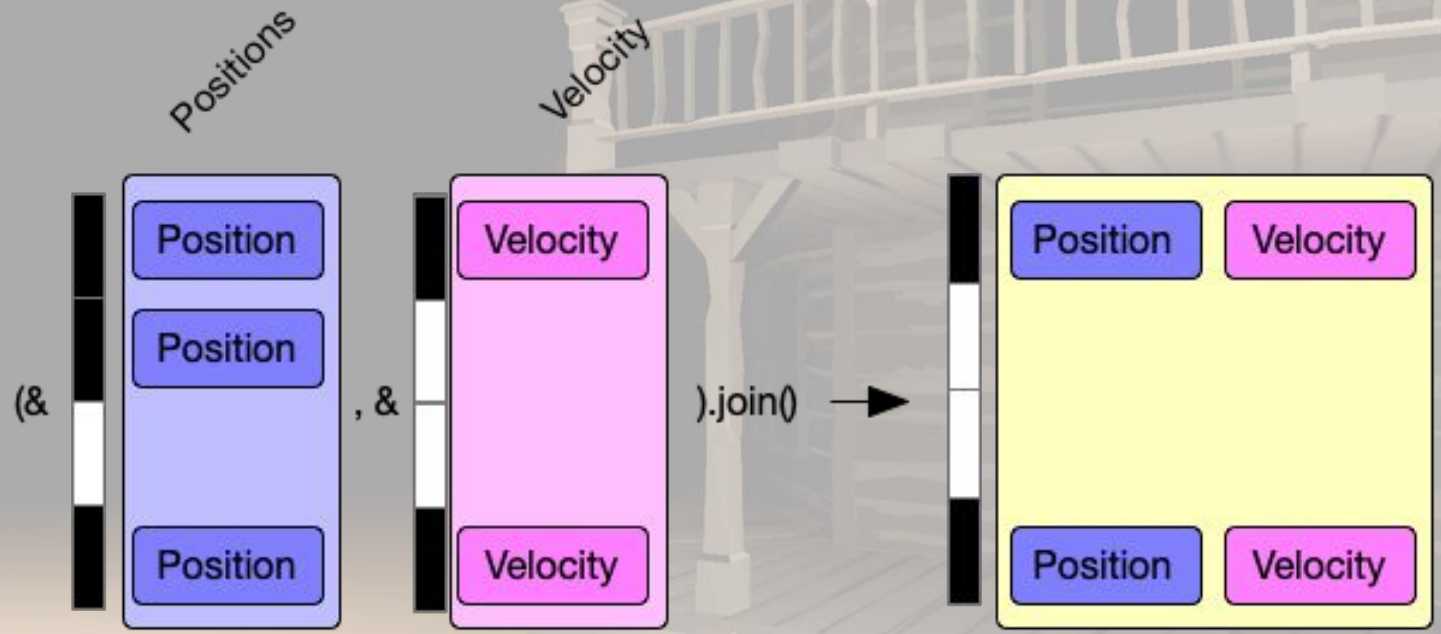
- ECS : Clément
- Physique : Alexandre (& Maÿlis)
- Graphique : Clément (& Barbara, Maÿlis)
- Son : Barbara
- GUI : Sébastien & Aymeric (& Nicolas, Amaury)
- Jeu test : Clément & Maÿlis
- Gestion du Git et correctifs : Amaury

III.2. ECS

1. Choix des librairies

SPECS vs LEGIONS

2. Explications



III.3. Graphics

1. Choix des librairies

- Glium → Wrapper Safe d'OpenGL

- Wavefront → Fichiers 3D pour nos objets

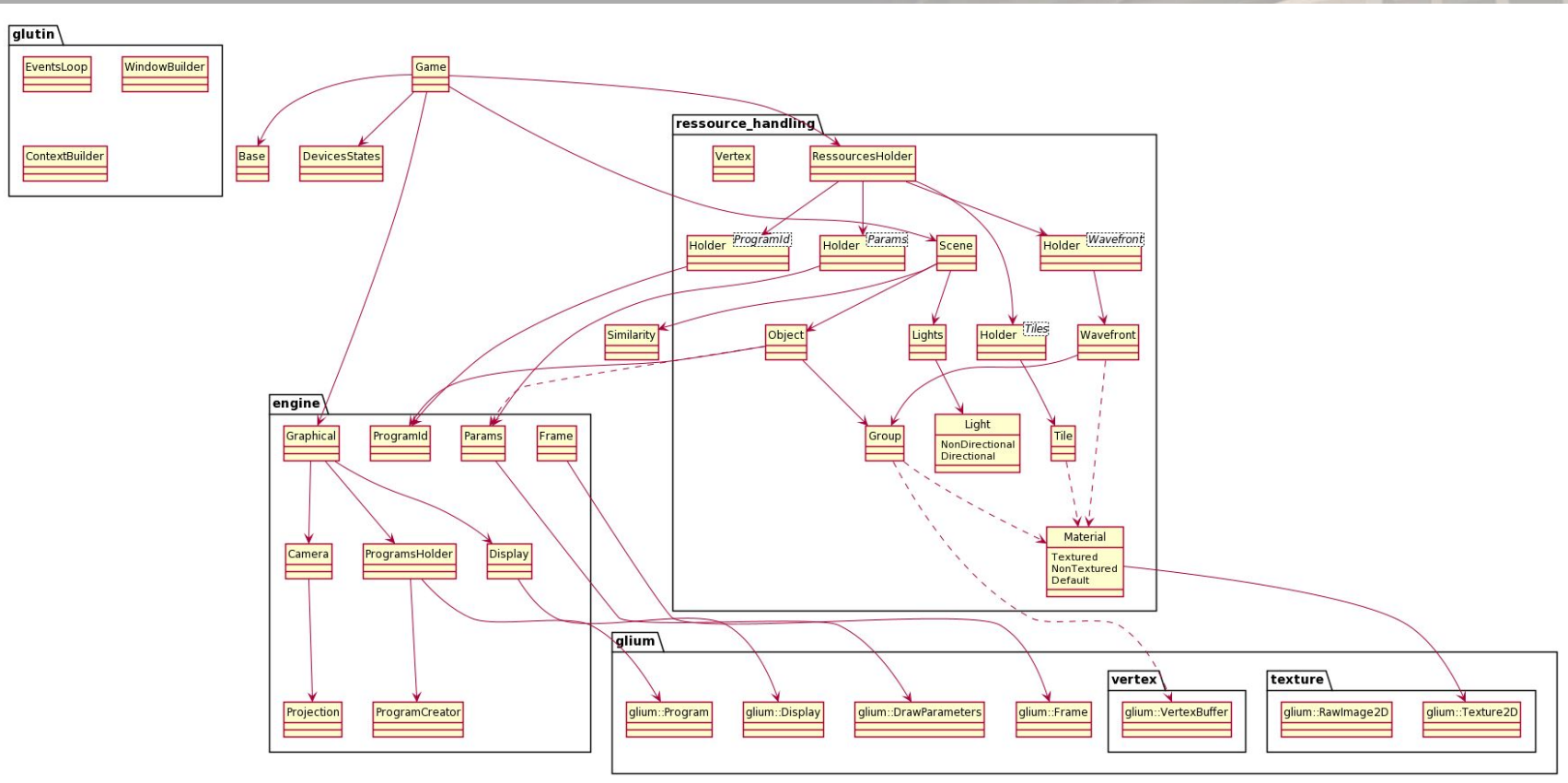
2. Explications

La partie graphique permet de :

- Projeter une vue sur la scène 3D en 2D
- Profondeur, objets, lumières et ombres

Elle est constituée de deux parties :

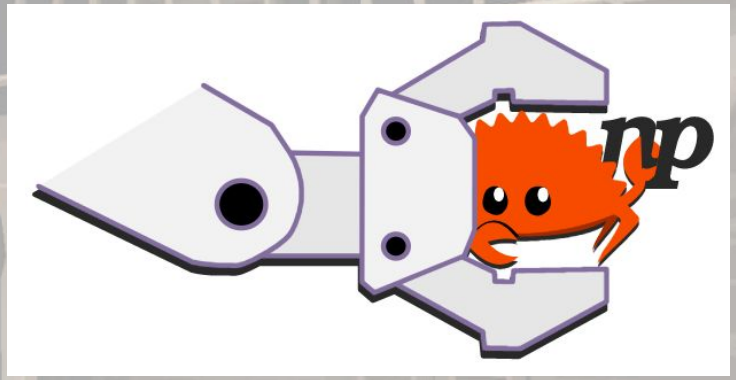
- Processing : Importe objets et crée structures de données
- Engine : Moteur de rendu


glutin
EventsLoop
WindowBuilder
ContextBuilder
Game
Base
DevicesStates
ressource_handling
Vertex
RessourcesHolder
Holder ProgramId
Holder Params
Scene
Holder Wavefront
Similarity
Object
Lights
Holder Tiles
Wavefront
engine
Graphical
ProgramId
Params
Frame
Group
Light
NonDirectional
Directional
Tile
Camera
ProgramsHolder
Display
Material
Textured
NonTextured
Default
Projection
ProgramCreator
glium
glium::Program
glium::Display
glium::DrawParameters
glium::Frame
vertex
glium::VertexBuffer
texture
glium::RawImage2D
glium::Texture2D

III.4. Physics

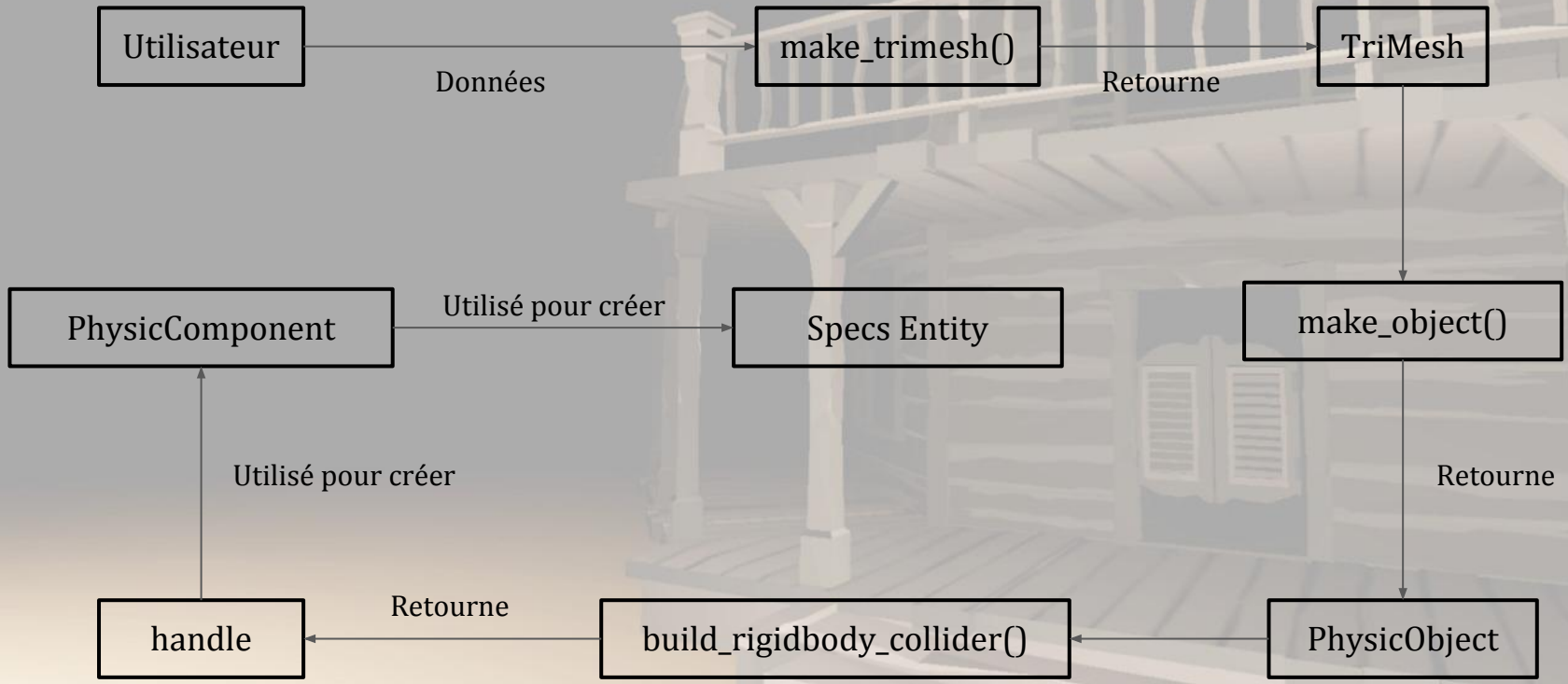
1. Choix des librairies

- Pas de binding de Bullet à jour
- Amethyst utilise nphysics
- Performances proches de celles de Bullet



ncollide

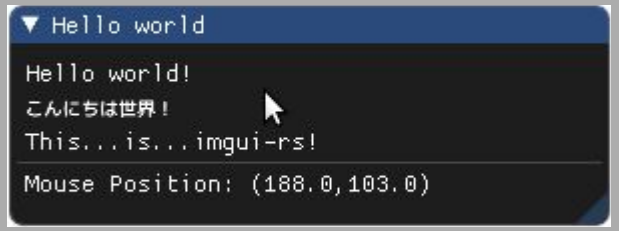
2. Explications



III.5. Graphical User Interface

1. Choix des librairies

Imgui-rs de Gekkio



- Une des librairies les plus documentées pour un gui en rust
- Utilise glium et winit
- Nécessite une structure similaire à Game

2. Explications

- Utilisation d’un moteur de rendu glium
- Utilisation d’une plateforme backend winit
    -> Création de fenêtre flottante avec Window::new

Apporte des éléments pré-faits :

- boutons
- barre de progression
- en-tête réduite
- etc.

III.6. Son

1. Choix des librairies

Ears -> basée sur OpenAl

- bien documentée,
- simple pour jouer des sons (%rg3d_sound)

principaux formats supportés:

- Wav
- Flac
- Vorbis

2. Explications

2 structures:

- SoundRessource basée ears::SoundData
  -> échantillon créé à partir d'un fichier son

- OneSound basée sur ears::Sound
  -> permet de jouer un son
  -> créé à partir d'un SoundRessource

Gestion des appels pour jouer un son par des gameEvents.

5 events: Option: placer spatialement le son

1. lecture en entier
2. lecture x sec
3. lecture à l'infini
4. Augmenter le volume sonore du jeu
5. Diminuer le volume sonore du jeu

III.7. Jeu test

Scénario : Découverte d’une maison autour d’une ambiance western

Gameplay : Se déplacer + ambiance western

Outils utilisés :

- Objets : recherche de modèles 3D + utilisation de Blender
- Sources de lumières : bougies, chandelier
- Menu : Revenir au jeu / quitter
- Sons : thème principal

IV. Pistes d’approfondissement

Optimisations

Rendre plus de choses customisables

Corriger la physique

Séparer le rendu sur son propre thread

V. Démonstration