POLYTECH LILLE

Université Lille1 Sciences et Technologies

Compte rendu de projet

SIMULATION 3D

LEULIET Nicolas
ALEXANDRE Nicolas

Sommaire

Introduction

L'objectif du projet de ce semestre est de réaliser une application permettant une visualisation 3D d'un système physique, ce projet a été proposé par Monsieur B. Conrard.

Pour cela nous utiliserons des logiciels de modélisation 3D (Unity, 3DSMAX) dans le but de créer une application ressemblant au palettiseur utilisé lors des séances de TP du premier semestre (cours de Grafcet).

Dans un premier temps nous argumenterons sur le choix du logiciel de modélisation en fonction des avantages et des inconvénients de chacun d'eux.

Par la suite nous détaillerons notre travail réalisé pour aboutir à la réalisation du palettiseur : la conception des éléments physiques, la création des différents scripts ainsi que le travail effectué pour obtenir un rendu visuel satisfaisant.

Avant de conclure nous reviendrons sur les différents obstacles auxquels nous nous sommes confrontés.

I. - Choix des logiciels

N'ayant jamais réalisé de modélisation 3D au cours de nos études, la première chose à faire a été de nous tourner vers notre enseignant encadrant (Mr Conrard) qui nous a dirigé vers trois différents logiciels : OpengGL, Unity 3D et Irrlicht. Pour chacun d'eux il existe une version gratuite accessible aux étudiant, il fallait donc faire un choix. Pour cela nous avons effectué une recherche pour chaque logiciel afin de déterminer lequel d'entre eux conviendrait le mieux à notre projet.

a) OpenGL

OpengGL est une spécification reconnu et très largement répandu dans le domaine des jeux vidéos professionnels, elle a été crée par la *Silicon Graphic.*

Sa documentation fournie par le développeur est bien complète et les applications crées sont « puissante », ce qui n'est pas forcément essentiel pour notre projet. De plus il n'existe pas d'environnement 3D pour créer les objets de base à la main (comprendre grâce à la souris), en effet tous les objets doivent être codés via des fonctions fournies par OpenGL.

Sur internet la communauté OpenGL est active mais elle est bien plus dirigée vers les professionnels, peu d'amateur l'utilise.

b) Unity 3D

Ce logiciel développé par *Unity Technologies* est ce qu'on appelle un « game engine » en d'autre terme il s'agit d'un moteur 3D et physique conçu pour la réalisation des jeux vidéo et des animations en temps réel.

Un petit détour sur leur site internet nous montre que ce logiciel est très utilisé par les éditeurs indépendants de jeux vidéos, la communauté y est très active et concerne les utilisateurs débutant.

Le logiciel Unity 3D propose un environnement 3D dans lequel il est possible de créer des objets complexes à partir d'objets basiques tels que des cubes, sphères, cylindres ou plan. Leurs animations sont ensuite réalisées dans des scripts attachés à l'objet en question, ceux-ci sont rédigé en C#, Javascript ou BOO.

c) Irrlicht

Irrlicht est un logiciel 3D open source programmé en C++, il n'a pas vocation a devenir lucratif, pour cette raison il est disponible gratuitement en téléchargement. Il est destiné en particulier aux amateurs souhaitant créer des jeux vidéo. Le logiciel est moins puissant que les deux cités précédemment, il est plus utilisé pour réaliser des environnement que pour des objets animés.

Irrlicht étant un logiciel peu connu il est difficile de trouver des tutoriels où des guides pour sa prise en main, même si la communauté anglophone possède un forum destiné à ceux souhaitant trouver de l'aide pour le logiciel.

Notre choix pour le projet s'est porté sur Unity 3D. Après discussion ce logiciel nous semblait le plus simple à prendre en main grâce notamment aux nombreux tutoriels disponibles pour apprendre à utiliser le logiciel. Il est aussi assez puissant et permet de réaliser tous les éléments nécessaires au projet.

II. - Présentation de l'environnement de Unity 3D

Après avoir obtenu une licence de Unity le logiciel est utilisable et à son ouverture on se retrouve face à l'environnement de travail suivant :

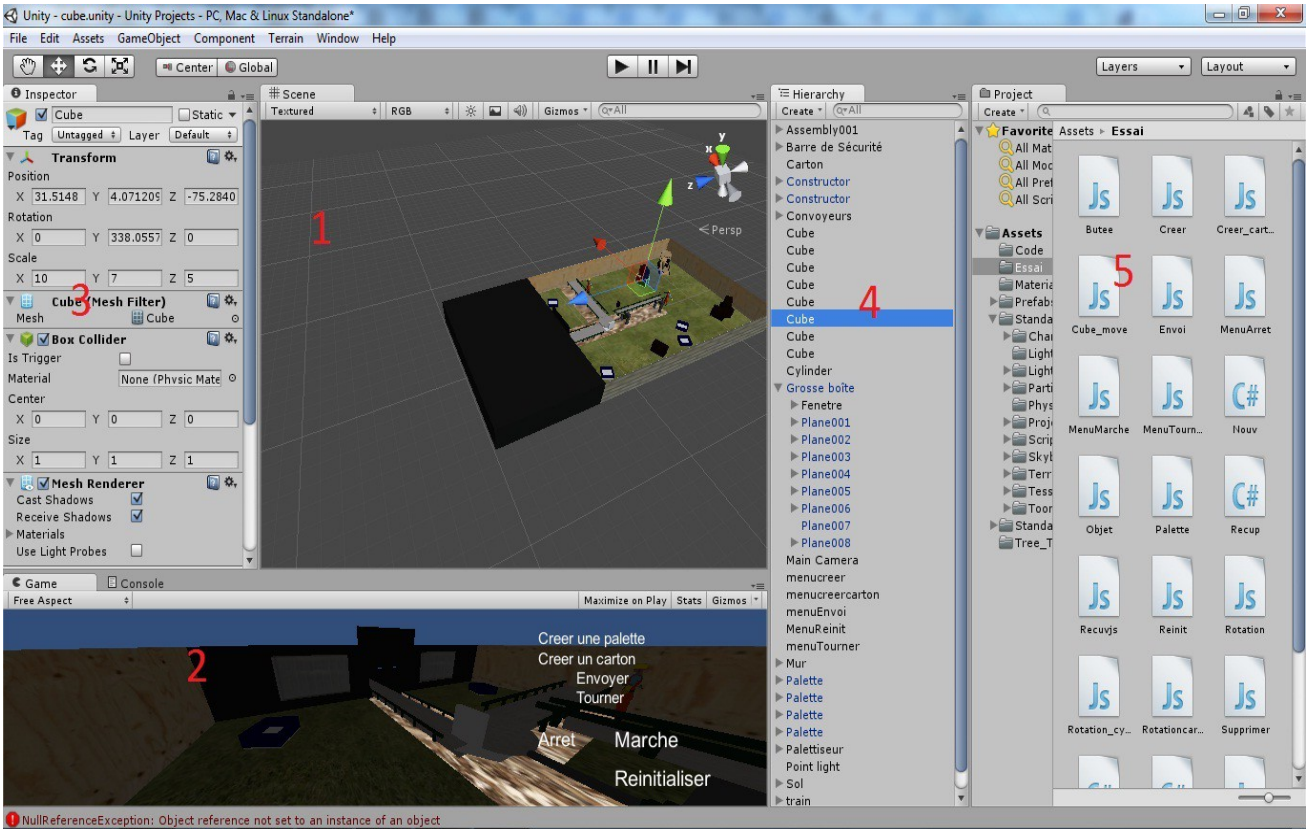

Figure 1 : Environnement de travail de Unity 3D

Comme on peut le voir, cet environnement est composé de cinq zones bien distinctes.

La première correspond à l'aspect visuel de la scène en trois dimension, on peut y placer directement les objets, modifier leur taille ou leur rotation grâce à la souris. On peut aussi naviguer dans cet environnement grâce aux touches directionnelles du clavier et avec la souris.

La seconde zone correspond au point de vue de l'objet « Main Camera », il s'agit en fait du point de vu que l'utilisateur aura quand il exécutera l'application. Grâce à un script conçu par nos soin, on est capable de se déplacer dans la scène une fois l'application lancée.

La troisième zone correspond aux caractéristiques d'un objet lorsqu'il est sélectionné, on peut y voir sa position dans le repère (x,y,z), sa rotation par rapport aux axes du repère, sa taille et les différents composants liés à cet objet (scripts, collision, gestion physique de l'objet, etc …).

La quatrième zone est l'endroit où l'on peut sélectionner les objets déjà créés et où l'on peut regrouper des objets afin de réaliser des « prefab », ensemble d'objets primitifs (cube, sphere, plan, cylindre) où objets 3D importés à partir d'un logiciel tiers.

Enfin la cinquième et dernière zone est un navigateur permettant de retrouver les scripts, les prefabs et les textures liés au projet ouvert.

III. - Présentation de 3DsMax

a) Description

3D Studio Max (ou 3Ds max) est un logiciel de modélisation et d'animation 3D, développé par la société Autodesk. Il est, avec Maya, Softimage XSI, Lightwave, Houdini et Blender, l'un des logiciels les plus utilisé dans le domaine de la simulation 3D.

Ce logiciel est issu du programme 3D Studio. Les programmeurs de Kinetis (une des divisions d'Autodesk Media and Entertainment) l'ont créé après l'avoir totalement repensé et donc créé quelque chose de nouveau.

Le logiciel 3ds max s'est rapidement fait connaître et est utilisé principalement dans le domaine du jeu vidéo. Il a également été utilisé dans d'autres domaines, notamment le film d'animation avec Kaena, la prophétie. 3D Studio Max est également utilisé dans de nombreux films comme: X-Men II, Bulletproof Monk, The Core, Final Destination II, Jason vs.Freddy ou plus récemment pour le film 2012.

Le choix de ce logiciel a été motivé par le fait qu'il nous a permis de modéliser les objets que l'on voulait inclure dans notre animation, mais surtout par le fait que chaque objet pouvait être exporté et que l'exportation est compatible avec Unity 3D, qui permettait également de modéliser des objets mais de manière plus complexe.

Ainsi, après une brève prise en main, la création de chaque élément a été faite plutôt rapidement et facilement.

b) L'application

Pour notre application (la simulation 3D d'un système palettiseur), plusieurs éléments devait être présent comme :

- Les tapis
- Les barres de sécurité pour les tapis
- Les cartons
- Les palettes
- La petite salle où partiront les palettes en fin de parcours
- Le vérin

Dans l'optique de travailler également sur le TP utilisant des trains, nous avons aussi créé un train (basique) que nous avons inclus dans notre animation à la fin (en tant que « décoration »).

L'image ci dessous nous montre un exemple pour le palettiseur (qui a été ensuite modifié dans Unity3D) et le train, dans lequel on peut voir l'espace de travail que nous propose le logiciel.

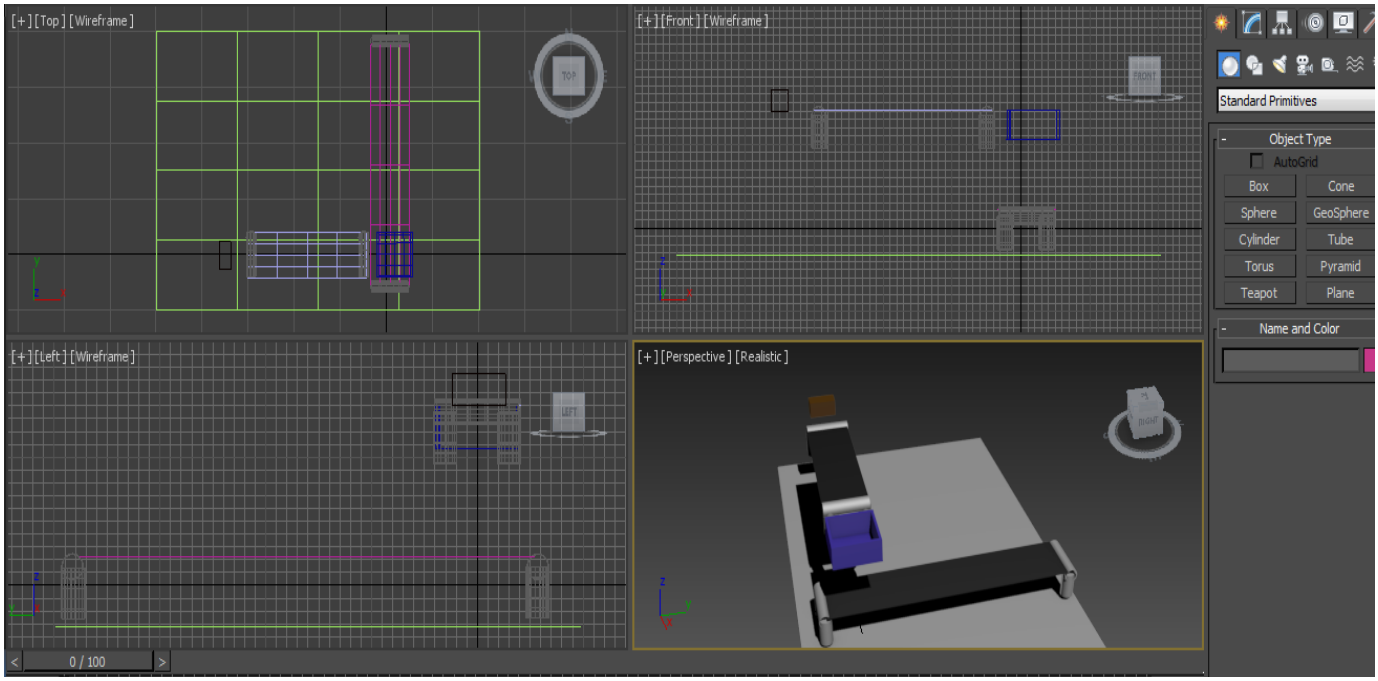

Figure 2 : Environnement de travail 3DS Max

Dans cet image, on peut observer en bas à droite l'ensemble en 3D alors que dans les autres fenêtre on peut avoir les objets vu de face, de gauche ou encore une vue de haut. Ainsi, on peut voir les deux tapis (légèrement modifiés sous Unity, la palette en bout du tapis qui va recevoir les cartons (boîte marron)).

La partie à droite de l'écran permet de créer d'un objet comme un cylindre, une boîte, un tube ou encore un simple plan.

IV. - Réalisation du projet

La prise en main du logiciel a été la première étape du projet, cela a été facilité grâce au nombreux tutoriels disponibles sur la toile. Il en existe sur des sites de vidéos en ligne tels que *Youtube* ou *Dailymotion*, mais aussi sur des sites internet plus spécialisés tel que *unity3d-france.fr* ou encore directement sur le site officiel du logiciel : *unity.com*.

Ces tutoriels nous ont permis de comprendre le fonctionnement des scripts, de créer des objets et de les animer.

a) Description des différents éléments et fonctions

Envoie de message par réseau UDP : deux scripts permettent l'envoie et la réception de message sur le réseau UDP, ils se nomment « UDPReceive.cs » et « UDPSend.cs », ceux-ci sont fournis librement sur le forum officiel de Unity 3D. Lors du lancement de l'application il est nécessaire de renseigner l'adresse réseau du PC « serveur ». On peut aussi envoyer des messages sur le réseau via un logiciel indépendant tel que « Engine Packet Builder » la réception sur Unity fonctionne toujours.

Convoyeurs (tapis roulants) : les tapis roulant sont constitués d'un cube allongé et de faible épaisseur, des barres de sécurité ont été mis tout du long des convoyeurs pour éviter que les cartons où les palettes ne tombent (des blocs de collider ont été mis en place sur ces barrières).

La friction physique du tapis est fixée à 0 afin de pouvoir faire glisser les objets dessus, les scripts liés à cet élément permettent d'arrêter le tapis ou de le mettre en marche. Pour faire bouger les objet sur le tapis on récupère la direction du tapis dans le script et on déplace l'objet qui est dessus dans la même direction.

Création d'objets : les palettes et les cartons peuvent être crées par l'utilisateur, il peut saisir s'il le souhaite les mots « Creer_carton » ou « Creer_palette » sur le réseau UDP, ou grâce aux menus placés au niveau de la camera. Dans tous les cas l'élément souhaité est crée aux coordonnés correspondants aux bouts des tapis roulant.

Rotation des cartons : lorsque le carton est placé dans la palette il est possible de lui faire effectuer une rotation de 90 degrés (selon l'axe Z). Cela peut être effectuer avec la commande « Tourner » envoyé sur le réseau ou par le menu en cliquant sur « Tourner ».

Expédier la palette : si l'utilisateur le souhaite il peut expédier la palette sur le dernier tapis roulant pour qu'il aille finir sa course, la commande « Envoyer » permet de réaliser cette fonction (aussi disponible sur le menu), le vérin ayant deux positions il est nécessaire d'effectuer la

commande une seconde fois pour que la plate-forme de chargement revienne en place (translation avant et arrière sur l'axe X).

Réinitialiser la scène : pour réinitialiser la scène l'utilisateur peut saisir « Relancer » dans le champs d'envoi sur le réseau UDP ou bien cliquer sur « Réinitialiser » dans le menu. Cela aura pour effet de relancer la scène et de faire disparaître tous les nouveaux objets créés.

b) Exemple de fonctionnement de script

Afin d'illustrer le fonctionnement des scripts réalisés sous Unity, nous allons prendre deux exemples fournis en annexe. Le premier sera le script de déplacement de la caméra (objet.js) et le second étant la création de carton (Creer_carton.js).

```
1  var moveSpeed = 1.0;
2  var turnSpeed = 1.0;
3
4  function Update () {
5       if(Input.GetButtonDown("Jump"))
6      {
7          transform.position.z += 1.0;
8      }
9
10     if(Input.GetButton("Forward")){
11         transform.position += transform.forward * moveSpeed * Time.deltaTime;
12     }
13     if(Input.GetButton("Backward")){
14         transform.position += -transform.forward * moveSpeed *Time.deltaTime;
15     }
16     if(Input.GetButton("Left")){
17         transform.eulerAngles.y += -turnSpeed * Time.deltaTime;
18     }
19     if(Input.GetButton("Right")){
20         transform.eulerAngles.y += turnSpeed * Time.deltaTime;
21     }
22 }
```

Figure 3 : Objet.js

Les variables *moveSpeed* et *turnSpeed* correspondent à la vitesse de la caméra en déplacement et en rotation, les actions qui se trouvent dans la fonction *Update* sont réalisées en continu (à chaque unité de temps, la fonction est parcourue).

Si la touche *Jump* est actionnée (= espace), la caméra se déplace de 1 sur l'axe Z. Si les touches *Forward (z)* ou *Backward(s)* sont actionnées alors la caméra ira vers l'avant (ou l'arrière), son déplacement sera proportionnel à la vitesse contenue dans la variable *moveSpeed*, le *Time.deltaTime* est une variable qui définie une unité de temps (pour effectuer le mouvement en continu).

Enfin un appuie sur la touche *Left(q)* ou *Right(d)* effectue une rotation autour de l'axe Y en fonction de la vitesse définie dans la variable *turnSpeed.*

```
var couleurEntrer : Color = Color.green;
var couleurSortie : Color = Color.white;
var Carton : GameObject;
var scriptB: UDPReceive ;
function Start(){
      scriptB = this.GetComponent("UDPReceive");
    }
function OnMouseEnter() {
    guiText.material.color = couleurEntrer;
}

function OnMouseExit() {
    guiText.material.color = couleurSortie;
    }
function Update(){
        if(scriptB.copie == "Creer_carton"){
                var clone_2 = Instantiate(Carton,Vector3(-0.1489229,13.24764,-62.2034),Quaternion.Euler(270, 0, 0));
                scriptB.strReceiveUDP = "";
                sens = 1;
            }
    }

function OnMouseUp() {
  Instantiate(Carton,Vector3(-0.1489229,13.24764,-62.2034),  Quaternion.Euler(270, 0, 0));

    }

```

Figure 4 : Creer_carton.js

Ce second script concerne la création des cartons, il est lié au menu « créer carton ». Dans un premier temps on défini deux variables de couleurs *couleurEntrer* et *couleurSortie*, ainsi qu'une variable d'objet *Carton* et un script correspondant au script de réception UDP.

Dans la fonction *Start(),* on lie la variable *scriptB* au script existant *UDPReceive.cs*, cela permet d'interagir avec les variables contenu dans ce script.

La fonction *OnMouseEnter()*, correspond à l'action qui se passe lorsque la souris passe sur le menu, ici on change simplement la couleur des lettres lorsque la souris passe dessus ou lorsqu'elle n'y est pas (*OnMouseExit()*).

Dans la fonction *Update(),* on scrute une copie du message reçu sur le port réseau, si ce message correspond à « Creer_carton » alors on effectue l'action qui consiste à créer un objet de type *Carton*, à l'emplacement (-0.1489229, 13.24764, -62.2034) du repère global et ayant une rotation sur l'axe X de 270 degrés. Ensuite on réinitialise le message reçu pour éviter les créations multiples.

Enfin la fonction *OnMouseUp()*, indique l'action à effectuer si l'on clique sur le menu, en l'occurrence ici on crée un objet de la même manière que pour l'action précédente.

c) Rendu visuel

Sous Unity 3D, le principal objectif était de rendre l'animation la plus esthétique possible en ajoutant des textures pour les objets, les murs, le sol etc. Aussi, le tapis servant à faire avancer les palettes et les cartons a dû être modéliser dans ce logiciel afin de pouvoir en faire une animation et rendre quelque chose de dynamique. Les textures n'ont pas été créé directement, certaines étaient déjà fournis avec le logiciel, d'autres ont été importé à partir de sites internet qui en proposaient (comme le site www.cgtextures.com par exemple).

Après les modifications effectuées et dans l'optique du meilleur rendu possible, nous avons obtenons la scène suivante :

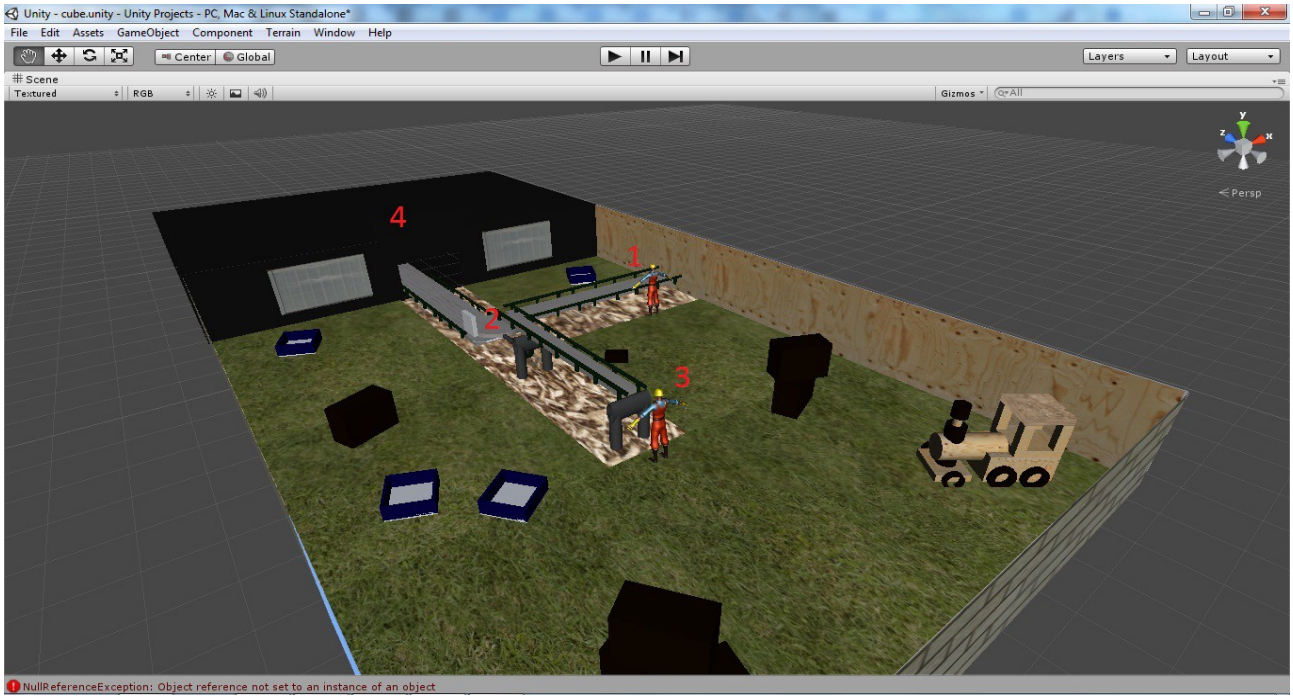

Figure 5 : Scène Unity 3D

d) Mis en place de la scène

L'animation va donc se décomposer en plusieurs étapes, comme suit :

- 1 : A ce niveau, nous allons créer la palette qui va servir à accueillir les cartons. Cette palette, à la demande de l'utilisateur (via UDP ou par appuie sur le bouton « Créer »), va apparaître et va avancer sur le tapis jusqu'à arriver au point 2.

- 2 : Cet élément va être l'endroit où la palette sera déposée afin d'attendre que les cartons arrivent. Chaque palette pourra accueillir deux cartons. Cet élément est tenu par un vérin qui laissera passer la palette une fois toutes les conditions réunies.

- 3 : Ici, et aussi à la demande de l'utilisateur, un carton sera créé et avancera sur son tapis. Le premier carton, une fois dans la palette, devra tourner avec celle-ci de 90 degrés puis attendre qu'un second carton vienne se déposer.

- 4 : Cet endroit servira à déposer les palettes munis de leurs cartons. En effet, une fois que les conditions seront réunies et que l'élément 2 aura relâché sa palette, celle ci avancera sur le dernier tapis et sera relâcher dans cette pièce.

On peut donc résumer le système par : en premier lieu, créer la palette et attendre qu'elle arrive jusqu'à l'élément 2. Une fois positionnée, un premier carton doit arriver (et faire une rotation de 90 degrés) dans la palette en attendant l'arrivée d'un second carton. Une fois la palette pleine, le vérin doit retirer l'élément 2 afin de laisser partir cette palette jusqu'à la pièce 4, où on peut imaginer que le carton sera traité par un second système.

V. - Ajout possible à l'application

A notre application, nous avions essayé d'apporter quelques fonctions supplémentaires que nous n'avons pas réussi à implanter :

a) Une pince

En effet, nous avions au début l'idée que lorsque les cartons seraient créés et déposés dans une palette, une pince (avec un ou plusieurs vérin) aurait été capable de prendre ces cartons et les déposer dans une seconde palette plus loin qui elle partirait dans la pièce de traitement des cartons. Le problème était dans l'accrochage des cartons avec la pince qui nous a été impossible à réaliser.

b) Capteurs

Afin de rendre ce système plus automatisé, il aurait pu être intéressant d'essayer de trouver un système avec des capteurs. Par exemple, un capteur de présence de palette pour dire aux cartons d'arriver ou encore un capteur disant que les cartons sont bien arrivés et que la palette peut partir. Cependant, ce système serait devenu très complexe et trop compliqué à mettre en place.

c) TP collisions des trains

Nous avons choisi de modéliser le palettiseur mais il était également possible de modéliser le TP de la ligne de train. Nous avons pour cela créer un train comme suit :

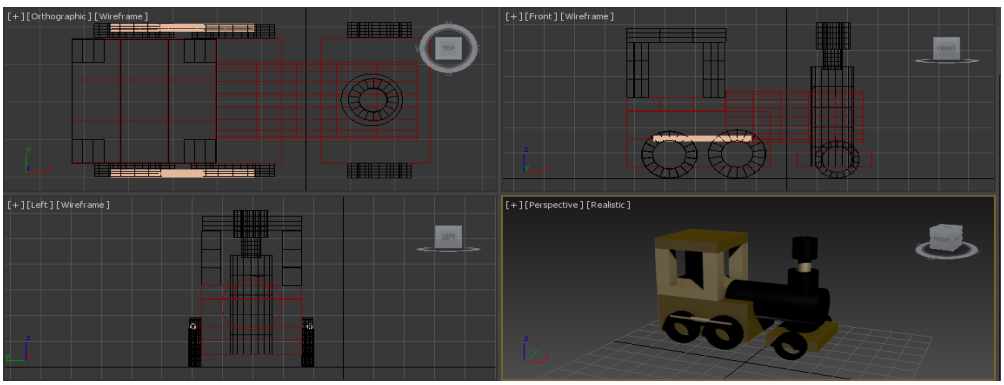

Figure 6 : Espace de travail 3DS Max

Ainsi, nous aurions pu trouver un système qui aurait pu simuler les trains avec des feux tricolores permettant de gérer les collisions entre ces trains. Cependant, nous avons juger que le palettiseur serait plus intéressant à réaliser et nous avons donc opté pour ce choix.

VII. - Mode d'emploi pour une utilisation normale

1. Créer une palette (« Creer_palette » sur réseau ou par le menu).
2. Attendre que celle-ci arrive sur la plateforme de chargement.
3. Créer un carton (« Creer_carton » ou par le menu).
4. Attendre que celui-ci atteigne la palette.
5. Tourner le carton (« Tourner » ou par le menu).
6. Créer un autre carton.
7. Attendre que celui-ci atteigne la palette.
8. Une fois dans la palettes, expédier le tout sur la fin de la chaîne (« Envoi » ou par le menu).
9. Ne pas oublier de replacer le vérin (« Envoi » une nouvelle fois ou par le menu).
10. Recommencer le cycle.

VIII. - Conclusion

Pour conclure, ce projet nous a permis de découvrir des logiciels d'animation 3D dont nous n'avions jamais eu l'occasion d'essayer. Ces logiciels et ces types de programmation étant utilisés dans de nombreuses applications, il a été intéressant d'avoir quelques connaissances supplémentaires dans ce domaine. Aussi, ce projet nous a permis de développer notre sens de l'organisation, notamment dans la répartition des tâches et la gestion du temps. C'était en effet notre premier véritable projet sur une période plutôt longue, tout cela nous a permis de comprendre un peu plus la manière de réaliser une application.

Le projet en lui même a été mené à bien et donne l'animation que nous avions imaginer au départ, malgrès quelques ajouts qui auraient pu être réalisés comme expliqué au dessus.

Cette application a pour but de simuler un palettiseur et pourrait être utilisée lors des TP en automatique, avec un système de capteur à ajouter afin de rendre cela programmable par grafcet par exemple. Le système reste plutôt facile et simple à comprendre mais il ne serait pas difficile d'ajouter des fonctions supplémentaires.

VI. - Annexes

Réception UDP

```
using UnityEngine;
using System.Collections;
using System;
using System.Net;
using System.Text;
using System.Net.Sockets;
using System.Threading;
using System.Net.NetworkInformation;

public class UDPReceive : MonoBehaviour {
   Thread receiveThread;
   UdpClient client;
   public int port = 26000;
    public string copie="";
   public static string strReceiveUDP="";
        string LocalIP = String.Empty;
   string hostname;

       public void Start()
   {
      Application.runInBackground = true;
      init();
   }
   // init
   private void init()
   {
      receiveThread = new Thread( new ThreadStart(ReceiveData));
      receiveThread.IsBackground = true;
      receiveThread.Start();
      hostname = Dns.GetHostName();
      IPAddress[] ips = Dns.GetHostAddresses(hostname);
      if (ips.Length > 0)
      {
         LocalIP = ips[0].ToString();
      }
   }

   void OnGUI()
   {
      Rect rectObj=new Rect(10,10,400,200);
      GUIStyle style = new GUIStyle();
      style.alignment = TextAnchor.UpperLeft;
      GUI.Box(rectObj,hostname+" MY IP : "+LocalIP+" : "+strReceiveUDP,style);
        copie=strReceiveUDP;
   }

   private  void ReceiveData()
   {
      client = new UdpClient(port);
      while (true)
      {
         try
         {
            IPEndPoint anyIP = new IPEndPoint(IPAddress.Broadcast, port);
            //IPEndPoint anyIP = new IPEndPoint(IPAddress.Any, 0);
            byte[] data = client.Receive(ref anyIP);
            string text = Encoding.UTF8.GetString(data);
            strReceiveUDP = text;
            //Debug.Log(strReceiveUDP);
         }
         catch (Exception err)
         {
            print(err.ToString());
         }
      }
   }

       public string UDPGetPacket()
   {
            return strReceiveUDP;
   }

   void OnDisable()
   {
      if ( receiveThread!= null)   receiveThread.Abort();
      client.Close();
   }
```

UDPSend.cs :

```
using UnityEngine;
using System.Collections;
using System;
using System.Text;
using System.Net;
using System.Net.Sockets;
using System.Threading;

public class UDPSend : MonoBehaviour
{
   public string IP = "192.168.1.10"; // default local
   public int port = 26000;
   IPEndPoint remoteEndPoint;
   UdpClient client;
   string strMessage="";

   public void Start()
   {
      init();
        }

       void OnGUI()
   {
      Rect rectObj=new Rect(40,120,200,400);
      GUIStyle style = new GUIStyle();
      style.alignment = TextAnchor.UpperLeft;
      GUI.Box(rectObj,"UDPSendData\n IP : "+IP+" Port : "+port,style);
      //
      strMessage=GUI.TextField(new Rect(40,160,140,40),strMessage);
      if (GUI.Button(new Rect(180,160,80,40),"Send"))
      {
         sendString(strMessage);
      }
   }

   public void init()
   {
      remoteEndPoint = new IPEndPoint(IPAddress.Broadcast, port);
      //remoteEndPoint = new IPEndPoint(IPAddress.Any, port);
      //remoteEndPoint = new IPEndPoint(IPAddress.Parse(IP), port);
      client = new UdpClient();
   }

   private void sendString(string message)
   {
      try
      {
         byte[] data = Encoding.UTF8.GetBytes(message);
         client.Send(data, data.Length, remoteEndPoint);
      }

      catch (Exception err)
      {
         print(err.ToString());
      }
   }

       void OnDisable()
   {
      if ( client!= null)   client.Close();
   }
}

```

Menuarret.js :

```
var couleurEntrer : Color = Color.green;
var couleurSortie : Color = Color.white;
var Obj : GameObject;
var sens : int;
var t : float;
var monScriptB : UDPReceive;
var scriptB: UDPReceive ;

function Start(){
  scriptB = this.GetComponent("UDPReceive");
  Obj = GameObject.Find("Conveyor");
    sens=0;
    }
function OnMouseEnter() {
    guiText.material.color = couleurEntrer;
}

function OnMouseExit() {
    guiText.material.color = couleurSortie;
  }

function OnMouseUp() {
    scriptB.strReceiveUDP = "Arret";
}
```

MenuMarche.js :

```
var couleurEntrer : Color = Color.green;
var couleurSortie : Color = Color.white;
var Obj : GameObject;
var sens : int;
var t : float;
var monScriptB : UDPReceive;
var scriptB: UDPReceive ;

function Start(){
  scriptB = this.GetComponent("UDPReceive");
  Obj = GameObject.Find("Conveyor");
    sens=0;
    }
function OnMouseEnter() {
    guiText.material.color = couleurEntrer;
}

function OnMouseExit() {
    guiText.material.color = couleurSortie;
  }

function OnMouseUp() {
    scriptB.strReceiveUDP = "Marche";
}
```

Creer_carton,js :

```
var couleurEntrer : Color = Color.green;
var couleurSortie : Color = Color.white;
var Palette : GameObject;
var Carton : GameObject;
var msg : String;
//var monScriptB : UDPReceive;
var scriptB: UDPReceive ;
var sens : int;
function Start(){
      scriptB = this.GetComponent("UDPReceive");
      sens =0;
    }
function OnMouseEnter() {
    guiText.material.color = couleurEntrer;
}

function OnMouseExit() {
    guiText.material.color = couleurSortie;
    }
function Update(){
        if(scriptB.copie == "Creer_carton"){
            if(sens == 0){
                 var clone_2 = Instantiate(Carton,Vector3(-0.1489229,13.24764,-62.2034),Quaternion.Euler(270, 0, 0));
                 scriptB.strReceiveUDP = "";
                 sens = 1;
            }
         else{
             var clone_3 = Instantiate(Carton,Vector3(-0.1489229,13.24764,-62.2034),  Quaternion.Euler(270, 0, 0));

             scriptB.strReceiveUDP = "";
             sens = 0;
         }
    }
}

function OnMouseUp() {

//cube = Instantiate(cube,transform.position+Vector3(0,0,4),Quaternion.identity);
  Instantiate(Carton,Vector3(-0.1489229,13.24764,-62.2034),  Quaternion.Euler(270, 0, 0));

    }
```

Reinit.js :

```
var couleurEntrer : Color = Color.green;
var couleurSortie : Color = Color.white;
var Palette : GameObject;
var Carton : GameObject;
var msg : String;
var scriptB: UDPReceive ;


function Start(){
    scriptB = this.GetComponent("UDPReceive");
    }

function OnMouseEnter() {
    guiText.material.color = couleurEntrer;
}

function OnMouseExit() {
    guiText.material.color = couleurSortie;
    }
function Update(){
    if(scriptB.copie=="Relancer"){
        Application.LoadLevel("cube");
    }
}

function OnMouseUp() {
        Application.LoadLevel("cube");

    }
```

MenuTourner.js :

```
var couleurEntrer : Color = Color.green;
var couleurSortie : Color = Color.white;
var Obj : GameObject;
var sens : int;
var t : float;
var monScriptB : UDPReceive;
var scriptB: UDPReceive ;

function Start(){
  scriptB = this.GetComponent("UDPReceive");
  //Obj = GameObject.Find("Carton(Clone)");
    sens=0;
    }
function OnMouseEnter() {
    guiText.material.color = couleurEntrer;
}

function OnMouseExit() {
    guiText.material.color = couleurSortie;
  }


function Update (){

}

function OnMouseUp() {

                    scriptB.strReceiveUDP = "Tourner";
}
```

Creer.js :

```
#pragma strict

var couleurEntrer : Color = Color.green;
var couleurSortie : Color = Color.white;
var Palette : GameObject;
var Carton : GameObject;
var msg : String;
//var monScriptB : UDPReceive;
var sens : int ;
var scriptB: UDPReceive ;
function Start()
{
    sens =0;
  scriptB = this.GetComponent("UDPReceive");
}

function OnMouseEnter() {
    guiText.material.color = couleurEntrer;
}

function OnMouseExit() {
    guiText.material.color = couleurSortie;
    }

function Update (){

    if(scriptB.copie == "Creer_palette"){
         var clone = Instantiate(Palette,Vector3(52.10294,19.2308,-13.97307),transform.rotation);
         scriptB.strReceiveUDP = "";
    }

}
function OnMouseUp() {

    //var cube : GameObject  = GameObject.Find;
//    var cube = GameObject.Find("cube");
//cube = Instantiate(cube,transform.position+Vector3(0,0,4),Quaternion.identity);
  var clone = Instantiate(Palette,Vector3(50.10294,19.2308,-13.97307),transform.rotation);

```

Objet.js (mouvement camera) :

```
var moveSpeed = 1.0;
var turnSpeed = 1.0;

function Update () {
     if(Input.GetButtonDown("Jump"))
    {
        transform.position.z += 1.0;
    }

    if(Input.GetButton("Forward")){
        transform.position += transform.forward * moveSpeed * Time.deltaTime;
    }
    if(Input.GetButton("Backward")){
        transform.position += -transform.forward * moveSpeed *Time.deltaTime;
    }
    if(Input.GetButton("Left")){
        transform.eulerAngles.y += -turnSpeed * Time.deltaTime;
    }
    if(Input.GetButton("Right")){
        transform.eulerAngles.y += turnSpeed * Time.deltaTime;
    }
}
```

Rotationcarton.js :

```
var scriptB: UDPReceive ;
var radius : float = 1;
var sens : int;

function Start(){
  scriptB = this.GetComponent("UDPReceive");
  sens=0;
}

function Update (){
        if(scriptB.copie == "Tourner"){
                if(sens == 0){

                  transform.Rotate(Vector3(0, 0, 90));
                        scriptB.strReceiveUDP = "";
                        sens=1;
                    }
                else{
                        transform.Rotate(Vector3(0, 0, -90));
                        scriptB.strReceiveUDP = "";
                        sens=0;
                    }
    }
}

```

Envoi.js :

```
var couleurEntrer : Color = Color.green;
var couleurSortie : Color = Color.white;
var Cyl : GameObject;
var sens : int;
var t : float;
var monScriptB : UDPReceive;
var scriptB: UDPReceive ;

function Start(){
  scriptB = this.GetComponent("UDPReceive");
    Cyl = GameObject.Find("Cylinder");
    sens=0;
    }
function OnMouseEnter() {
    guiText.material.color = couleurEntrer;
}

function OnMouseExit() {
    guiText.material.color = couleurSortie;
  }


function Update (){
        if(scriptB.copie == "Envoyer"){
                if(sens == 0){
                t += Time.deltaTime/3;

                  Cyl.transform.position = Vector3.Lerp(Vector3(0.39573,4.802913,-23.45374), Vector3(0.39573,5.102913,-14.36388), t);
                        scriptB.strReceiveUDP = "";
                        sens=1;
                    }
                else{
                        Cyl.transform.position = Vector3(0.39573,5.102913,-14.36388);
                        scriptB.strReceiveUDP = "";
                        sens=0;
                    }
}
function OnMouseUp() {

        if(sens == 0){
                t += Time.deltaTime/3;

                  Cyl.transform.position = Vector3.Lerp(Vector3(0.39573,4.802913,-23.45374), Vector3(0.39573,5.102913,-13.36388), t)
                        sens=1;
                    }
                else{
                        Cyl.transform.position = Vector3(0.39573,5.102913,-13.36388);
                        sens=0;
                    }
            }


```