Rapport de Stage : Jeu 3D Biodiversité Sur Table Tangible

Développement d'une API & un Jeu 3D Biodiversité Pour une Table Tangible. Développement d'une gestionnaire & un serveur de mise à jour pour une table Tangible.

Sous la direction de

M. Mourad YESSAAD & M. Martin Titillon

Table des matières

Table des Illustrations

Remerciement

Je remercie particulièrement mon maître de stage, Mourad Yessaad, responsable Mobile, 3D et vidéo pour m'avoir formé, suivi et encadré durant mon stage, ses conseils qui m'ont dirigé dans chaque étape de ce projet.

Je remercie M Saadi Ben Saïdane, le directeur de l'entreprise « ColocArts » pour sa confiance en moi et son soutien durant mon stage.

Je remercie Martin Titillon, le développeur Unity/C#, pour m'avoir suivi et conseillé durant toutes les phases de développement.

Je remercie Anthony Amico, le graphiste de 2D/3D pour son soutien dans la partie graphique durant mon Stage.

Je remercie Angélique Tron, l'assistante de direction pour s'être occupée de mes démarches administratives.

Je remercie Christopher Paulandré, Chef de projet de Jeu Biodiversité sur la Table Tangible, pour m'avoir conseillé et suivi sur les besoins client durant toutes les phases de ce stage.

Et plus généralement je remercie tout le reste de mes amis au sein de« ColocArts » : Lionel Ippolitto, Jimmy Teffit, Julie Detrailles, Caroline Klein, James Choux, Thomas Titillon, Pinelli Luc, Jocelyn Barral-Baron…pour m'avoir accueilli et supporté durant mon stage.

Enfin, Je remercie Tous Mes enseignants du Master WIC à l'Université Pierre-Mendès-France pour m'avoir appris les bases qui m'ont été utiles pour les différentes tâches de mon stage. Et plus particulièrement M. Benoît LEMAIRE et Mme Julie Dugdale pour m'avoir admis à l'inscription dans ce master et pour leur soutien durant ma scolarité.

Résumé

Mon stage au sein de l'entreprise « ColocArts » se divise en 3 grandes parties, et chaque partie contient Deux phases principales (Phase de Conception et phase de développement).

La première partie de mon stage concerne la conception et le développement d'une bibliothèque pour la communication avec la nouvelle version de la table tangible.

Une deuxième partie consiste à faire la conception du jeu biodiversité avec le développement en utilisant Unity 3D et c# comme langage de programmation.

Une troisième partie englobe la conception et le développement d'un gestionnaire à la table qui gère les différentes applications de cette dernière ainsi qu'une application« serveur » qui sert à mettre à jour les applications de la table à distance. Les deux applications ont été développées avec c#.

Mot clés : Jeu, Unity 3D, C#, Programmation réseau, Table Tangible, l'identification par radiofréquence.

Abstract

My internship inside the company “Colocarts” was divided over 3 main parts, and each part contains two main phases (design and development).

The first part of this internship requires the design and development of a framework for communication with the new version of the tangible table.

A second part is to design the biodiversity game with development using Unity 3D and C # as the programming language.

A third part is to design and develop two application, both applications were developed with C #:

- Application “Manager” for the table that manages the applications for the table
- Application “Update Server” is used to update any application on the tangible table using the network.

Keywords: Game, Unity 3D, C #, Network Programming, Tangible Table, Framework, Radio Frequency IDentification.

Glossaire

API (Application Programming Interface) : une bibliothèque standardisée de programmes informatiques qui offre des services de communication avec un matériel ou logiciel, qui contient des ressources bien documentées telles que (classes, fonctions) pour une réutilisation facile par des différents développeurs informatiques.

Framework : un groupe de bibliothèques ou classes réutilisables par un logiciel ou un programme informatique (.Net par exemple).

RFID (Radio-frequency identification): transfert de données à distance en utilisant des champs électromagnétiques.

RFID Tag : sont des étiquettes qui contiennent des informations (identificateur et des données supplémentaires), ces étiquettes comprennent une antenne associée pour recevoir et émettre des requêtes radios avec des puces RFID.

RFID Sensor : sont des puces électromagnétiques qui servent à la détection des tags RFID et leurs informations, qui sont présents dans leurs zones de détection.

RFID Slab (Dalle en français) : un groupe de puces RFID liées ensemble par un seul émetteur Trame Tangible, et qui ont la même adresse IP.

LED (Light-Emitting Diode): en français diode électro-lumineuse, un composant électronique capable d'émettre de la lumière.

PARTIE I. Introduction :

1) Contexte :

Etant passionné par l'industrie des jeux vidéo et leurs développement, je me suis décidé depuis longtemps à travailler dans cette dernière industrie (Projet fin d'étude de Licence). Ce stage répond à mes attentes d'améliorer mes compétences dans ce domaine, qui est devenu aujourd'hui un domaine vaste et très productif (plus de 5 millions jeux sont produits) après l'immense progrès technologique des matériaux (mobiles, ordinateurs, consoles de jeux).

Dans le cadre du projet du « Contrat de diversification des stations durables (CDSD) – Programme opérationnel inter-régional du massif alpin » initié et animé par le Parc Naturel Régional du Vercors, nous prévoyons d'installer un espace d'information dédié à la plus grande réserve naturelle de France. Cette muséographie intérieure et extérieure viendra compléter le projet de réhabilitation de l'ancienne cure qui abrite aussi l'office de tourisme.

Suite au travail entre le CPIE Vercors (Centre Permanent d'Initiatives à l'Environnement) et Multicom, il a été mis en place un jeu « la biodiversité au bout des doigts » testé et manipulé par le grand public. Ce jeu répond aux messages que l'espace muséographique doit transmettre aux visiteurs.

Outil innovant mais encore expérimental, il a donc été proposé la mise en place de l'outil table interactive RFID « Tangisens » avec le scénario « biodiversité au bout des doigts » adapté à la biodiversité du Vercors.

2) Organisme d'accueil :

La société ColocArts, est une Agence Nouveaux Médias qui allie créativité graphique et développement informatique innovant sur le Web et en mobilité. Elle est située à Grenoble et a été créée en janvier 2010, son domaine d'activité est le Web, la création et le développement2D/3D, elle compte plus de 10 Employées.

Figure 1 : ColocArts Logo

L'entreprise divisée sur 3 grands axes d'activités : développement Web, création graphique, développement logiciel et jeux.

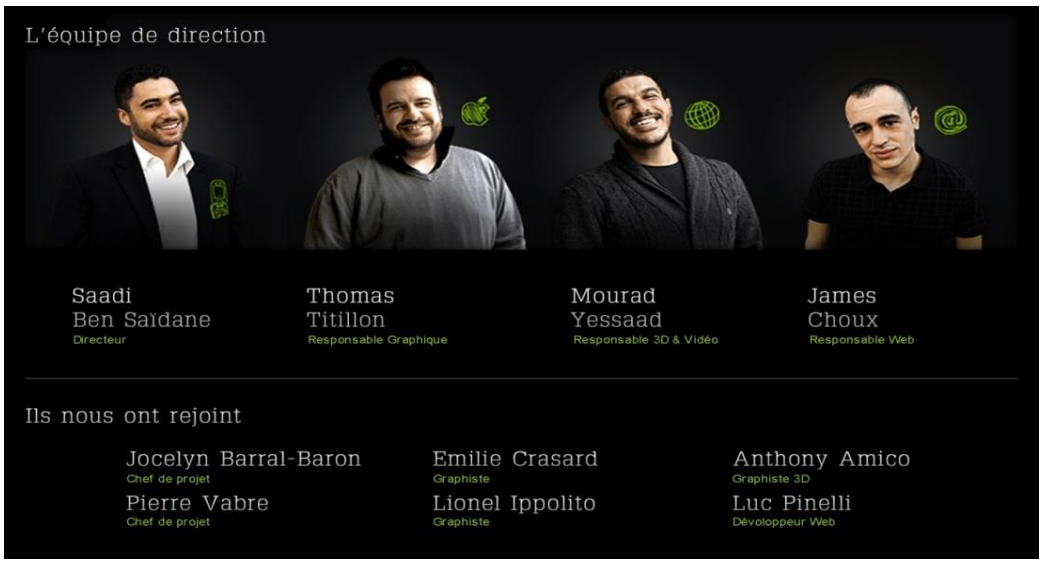


Figure 2 : les équipes de ColocArts

Aujourd'hui, ColocArts fourni des solutions multimédias riches (2d/3d /son/vidéo) sur le web, Mobile, Consoles de jeux ou ordinateurs pour ces différents clients :

- Stratégie et conseil en communication.
- Images de synthèse 3D et vidéo.
- Développement Web, logiciel et mobile.
- Conception graphique et d'identité visuelle.
- Création de concepts de communication originaux.
- Réalisation de plans de communication et de plans médias.

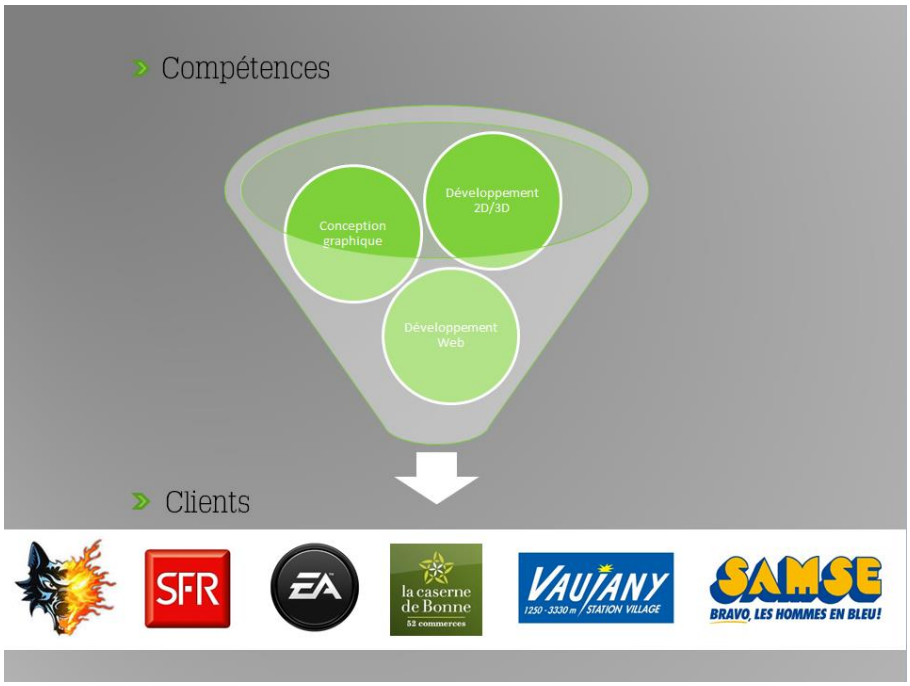


Figure 3 : Activités de ColocArts

PARTIE II. Etude de l'existant :

1) API Tangible Existante:

Premièrement il n'existe pas une API dans les sources que j'ai pour la réutiliser. Les classes et les fonctions qui établissent la communication avec la table sont mélangées directement avec l'ancienne version de jeux « La biodiversité au bout des doigts » qui est écrite en java (sous formes des packages).

Deuxièmement, l'ancienne bibliothèque était dédiée pour une version de la table à base de diodes lumineuses (comme montre la figure suivante).Au contraire, la nouvelle version de la table tangible est munie d'un écran LCD HD et d'une carte graphique (Nvidia):

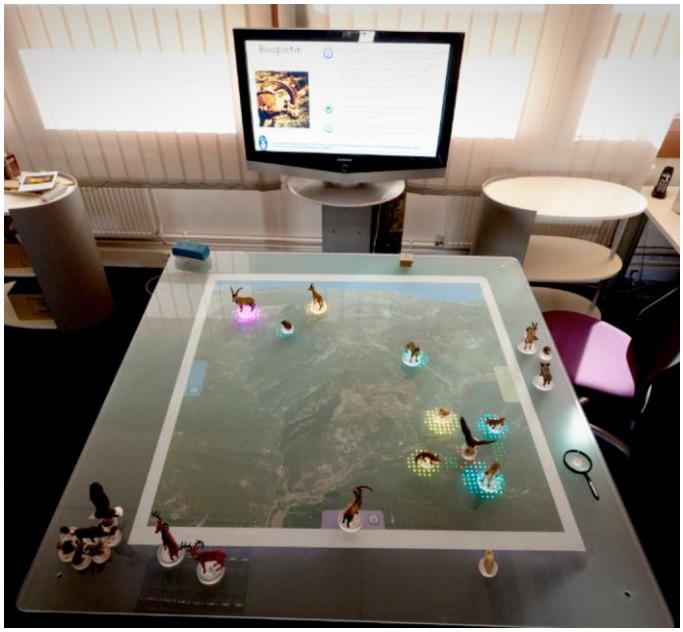


Figure 4 : Jeux biodiversité Ancienne version (A base des diodes Lumineuses)

2) Ancienne Version : « Biodiversité au bout des doigts »:

L'ancienne version de jeux de « La biodiversité au bout des doigts » se déroule sur La table interactive à base des diodes qui est placée face à un grand écran, avec un espace pour qu'un « maitre du jeu » puisse passer. Sont placées sur la table trois zones d'informations afin que 3 participants puissent jouer à tour de rôles.

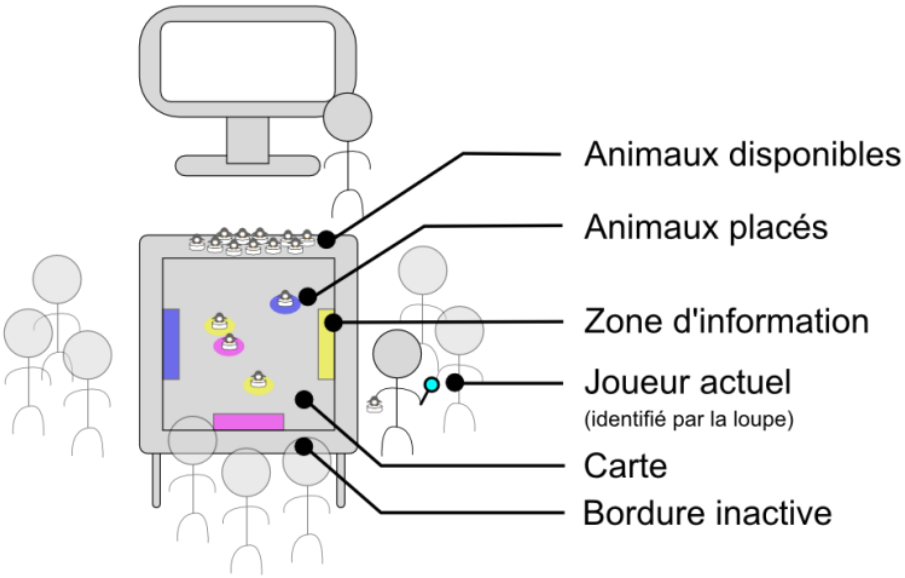


Figure 5 : Jeu Biodiversité Version Ancienne

Contrairement au scénario demandé par le client.

3) Conclusion :

D'après le cahier des charges signé entre «ColocArts » et la Mairie de « Gresse en Vercors » et l'étude de l'existant nous avons conclu :

1. Incompatibilité de l'API existante avec la version de la table et à nos futurs besoins avec la technologie Tangible.
2. Ambigüité des droits d'auteur des codes sources du jeu «Biodiversité au bout des doigts ».
3. Incompatibilité du scénario de « la Biodiversité au bout des doigts » avec le scénario désiré par La Mairie de Gresse de Vercors.
4. Pas d'écran externe pour l'affichage des informations (informations d'un animal et informations d'une zone).
5. Le scénario demandé par le client contient seulement deux joueurs qui jouent à tour de rôle (pas trois joueurs ou un joueur).
6. La nouvelle technologie de la table diffère de l'ancienne version et les diodes lumineuses ont été remplacées par un écran LCD.

PARTIE III. Travail Demandés :

1) Le travail à faire dans mon stage :

Au début il faut comprendre la démarche d'interaction tangible dans un contexte RFID (Radio Fréquence ID). Ensuite à cause d'incompatibilité du code source existant et son ambigüité des droits d'auteur et lors de la réunion de l'équipe de ce projet, je dois reconcevoir une nouvelle API qui répond à nos futurs besoins de la technologie tangible.

La tâche principale de mon stage est de concevoir et développer le jeu Biodiversité selon le Scénario dans le cahier des charges signé entre «ColocArts » et « la Mairie de Gresse en Vercors ».

Dans un deuxième temps il est primordial de concevoir et développer une interface de gestion des applications de la table tangible qui contient les fonctionnalités suivantes :

- Démarrage automatique des applications et son paramétrage à partir d'objets tangibles.
- Installer/désinstaller/mise à jour des applications de la table.

Comme une tâche optionnelle et pour une maintenance à distance de la table et ses futures applications, je dois concevoir et développer une application serveur de mise à jour des applications de la table.

PARTIE IV. Outils et Méthodologie:

1) Outils de développement :

a. Pourquoi c# ? :

J'ai choisi c# parce que premièrement je connais le langage depuis 5 ans et deuxièmement pour les raisons suivantes :

- Langage orienté objet très riche grâce aux plusieurs versions de framework « .Net ».
- Le langage de développement au sein de l'entreprise « ColocArts ».
- Le langage des scripts utilisés avec Unity 3D.

b. Microsoft « .Net » :

Une bibliothèque développée par Microsoft pour le développement logiciel sous Windows dans plusieurs langages (C, C++, C# et VB etc...). Plusieurs versions de « .Net » existent et la figure suivante montre la différence entre eux :

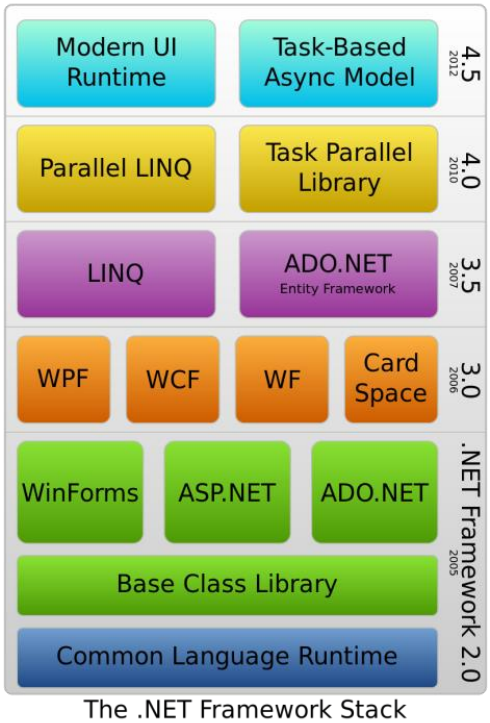


Figure 6 : différentes Versions du ".Net"

c. Unity 3D :

Unity est un moteur de jeu puissant, qui possède une variété d'outils qui peuvent être utilisés pour répondre à nos besoins spécifiques. L'éditeur de Unity est intuitif et personnalisable ce qui nous permet une plus grande liberté dans notre flux de travail. J'ai utilisé Unity 3D pour développer ce jeu, dans sa version professionnelle 4.3.1f et j'ai le choix entre programmer en c#, JavaScript ou Boo. C# a été choisi pour le langage utilisé avec Unity 3D car il est le langage principal utilisé au sein de l'entreprise « ColocArts » et pour les futures mises à jour des applications développées:

Figure 7 : Statistiques Unity 3D 2014

i. Unity 3D « Game-Object » :

C'est la classe de base pour toutes les entités dans des scènes Unity 3D. Les « Game-Objects » sont les objets fondamentaux dans Unity 3D qui représentent des personnages, accessoires et décors. Ils agissent comme des conteneurs de composants, qui mettent en œuvre la fonctionnalité réelle. Par exemple, un objet de lumière est créé par la fixation d'un composant de lumière à un« Game-Object ».

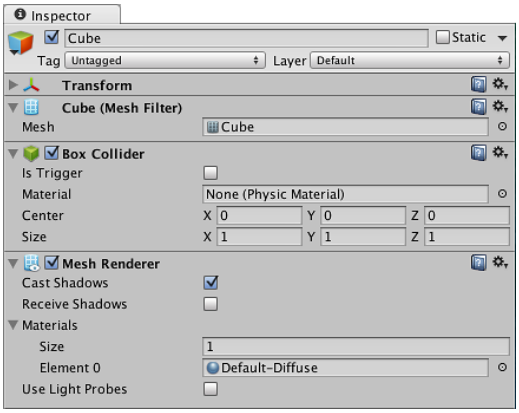


Figure 8 : Unity 3D Game Object

ii. Unity 3D Inspector:

Les jeux dans Unity 3D sont constitués de plusieurs « Game-Objects » qui contiennent des mailles, des scripts, des sons ou d'autres éléments graphiques comme des lumières. L'inspecteur affiche des informations détaillées sur notre « Game-Object » actuellement sélectionné, y compris tous ses composants et leurs propriétés. Ici, nous modifions les fonctionnalités de Game-Objects dans notre scène.

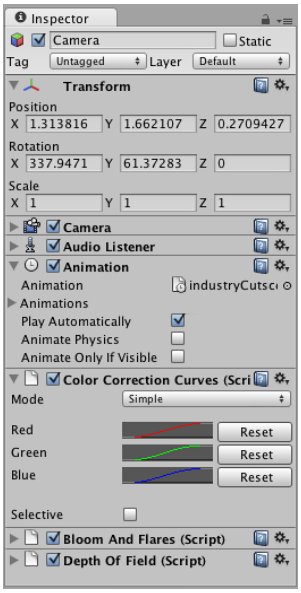


Figure 9 : Unity 3D Inpector

Toute propriété qui est affichée dans l'inspecteur peut être modifiée directement. Même les variables de script peuvent être modifiées sans changer le script lui-même. Nous pouvons utiliser l'inspecteur pour modifier les variables à l'exécution et de trouver le « Game-play » magique pour notre jeu.

Dans un script, si nous définissons une variable publique d'un type d'objet (comme Game-Object ou de transformation), nous pouvons glisser et déposer un « Game-Object » ou un « Prefab » dans l'inspecteur.

iii. Unity 3D Prefabs:

Unity 3D dispose d'un type « Prefab » qui nous permet de stocker un objet complet avec des composants et des propriétés. Les « Prefabs » nous permettent de créer de nouvelles instances d'objets dans la scène. Toutes les modifications apportées à un « Prefab » sont immédiatement répercutées dans toutes les instances de ce « «Prefab », mais nous pouvons également modifier les composants et les paramètres pour chaque instance individuellement.

d. Visual Studio 2013 :

J'ai utilisé Visual Studio dans sa version de 2013 pour le développement des deux applications « gestionnaire des applications » et « serveur de mise à jour» pour la table. Ce dernier est un environnement de développement de logiciels informatiques conçu par Microsoft, il se base sur plusieurs versions de Framework .Net (2.0, 3.0, 3.5 et la dernière version .Net 4.5), il représente un ensemble complet des outils de développement dans plusieurs langages (Visual Basic, c++, c#, j#), et j'ai choisi c# comme langage de programmation pour:

- La compréhension facile par l'équipe des développeurs au sein de « ColocArts ».
- La révision facile.

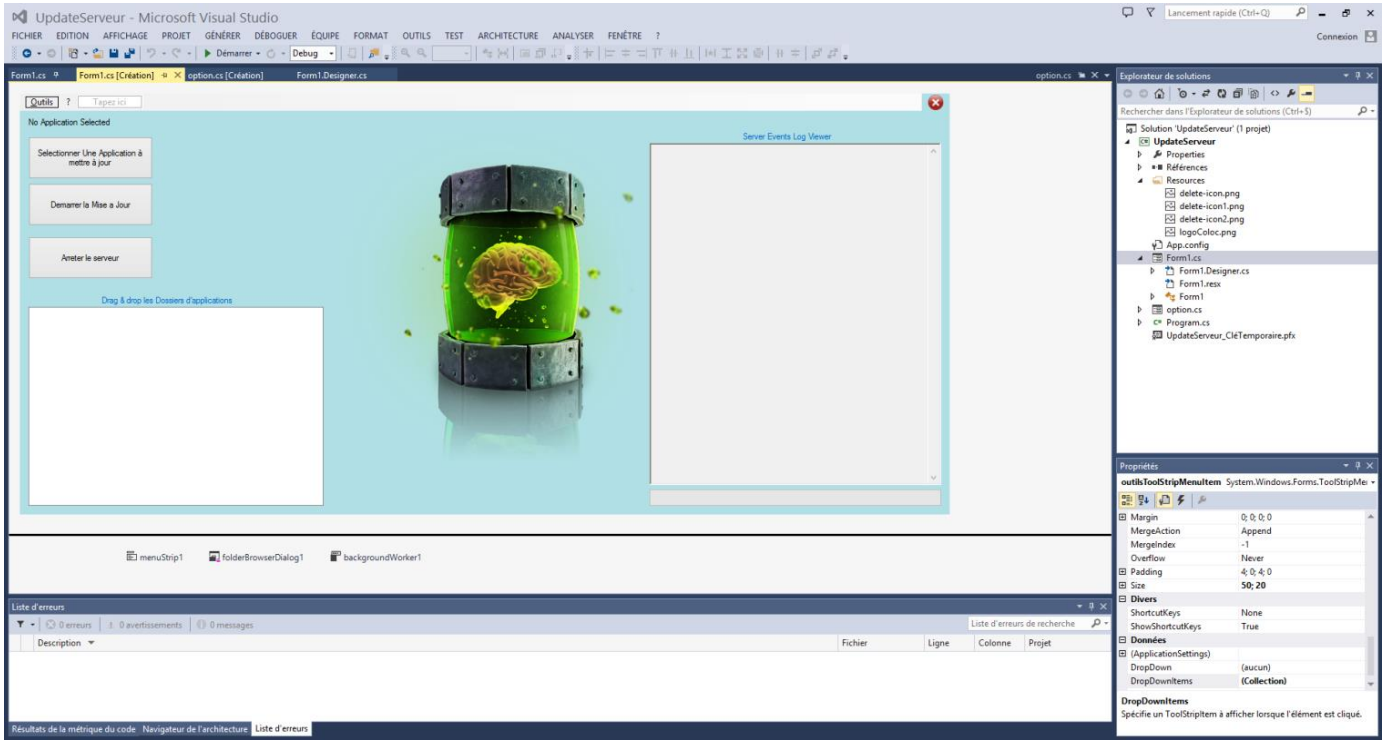


Figure 10 : Visual Studio, solution de Serveur de Mise à jour

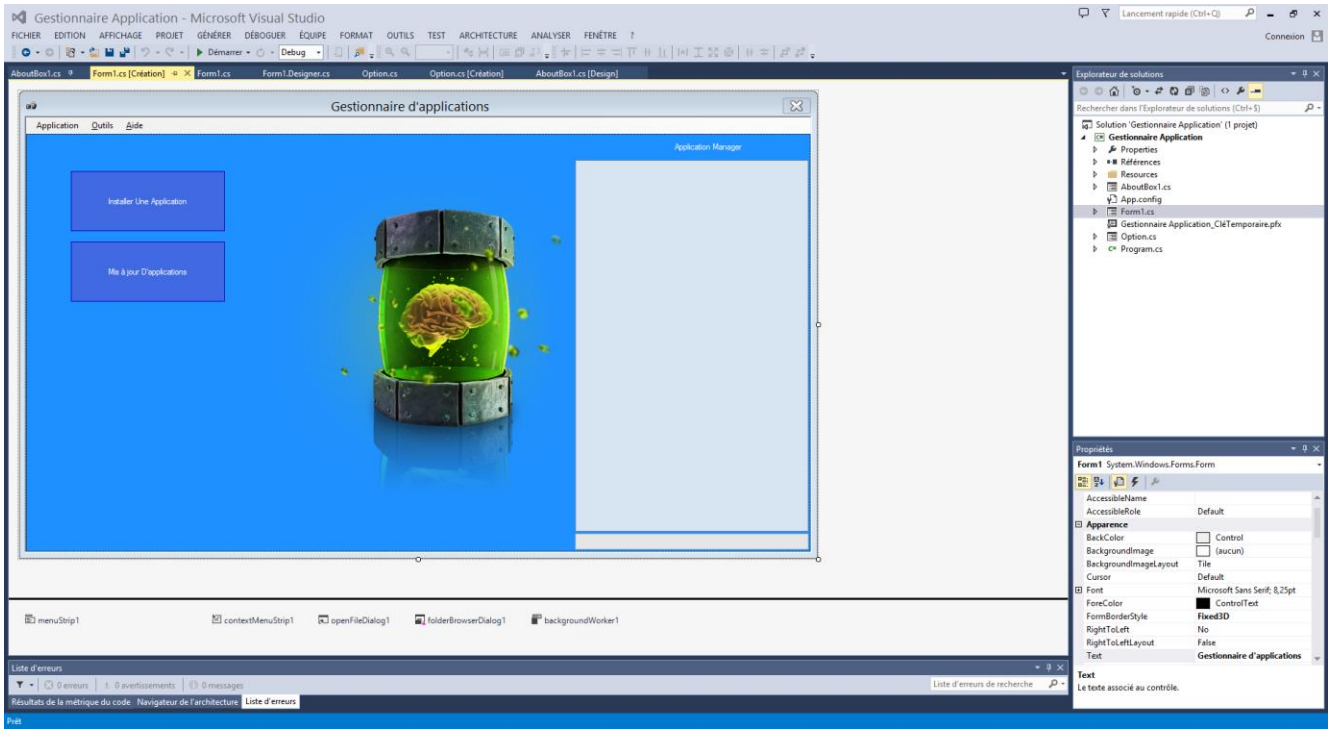


Figure 11 : Visual Studio, solution de Gestionnaire des applications

e. Autodesk « 3Ds Max » :

Comme il dit son nom, « 3Ds max » est un logiciel de modélisation et d'animation en 3D, développé par Autodesk (la version 2010 a été au cœur des Oscars cette année, avec plus de 11 films nominés employant Autodesk « 3Ds Max »). J'ai utilisé ce dernier pour créer nos différents composants

graphiques avec l'assistance d'Anthony (graphiste au sein de ColocArts), importés après dans Unity 3D pour les intégrer dans notre jeu. Par exemple, la définition des objets des zones sur la carte:

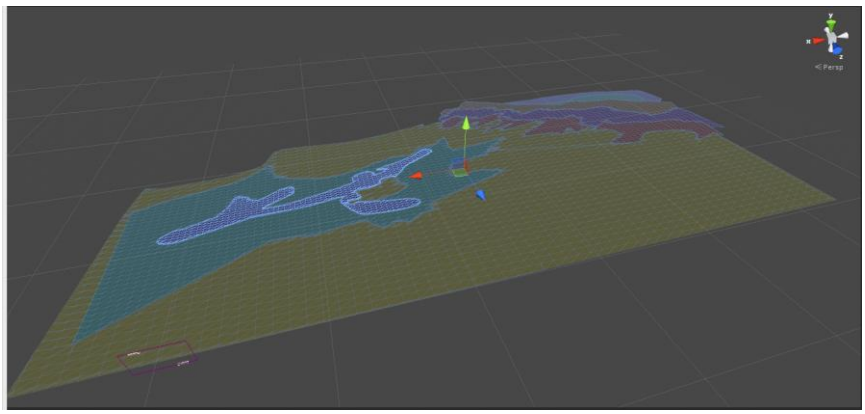


Figure 12 : Autodesk 3Ds Max : Objets Zones 3D

f. Logiciels d'Adobe:

Toutes les images au sein de nos applications sont créé par « Photoshop » ou « Illustrator », les deux logiciels sont développés par « Adobe » et offrent un taux de productivité énorme, j'ai utilisé la version CS6 de deux logiciels qui sont complémentaires :

i. Photoshop Cs6 :

Un logiciel de retouche, traitement et création d'image, il se base sur le traitement des images matricielles, il offre une grande galerie d'outils, d'effets et de filtres…

Figure 13 : Adobe Photoshop Cs6

ii. Illustrator Cs6 :

Un logiciel de création d'images vectorielles (des images indépendantes de la résolution, elles ne perdent pas de qualité si on les agrandit), complément de Photoshop.

Figure 14 : Adobe Illustrator Cs6

2) Outils de Conception :

J'ai utilisé « UML 2.0 » comme méthode de conception pour toutes nos applications développées, elle nous permet de faire collaborer tous les participants de ce projet (clients, chefs projets, programmeurs) et elle donne une modélisation forte de très haut niveau (indépendamment de programmation). Tous les diagrammes UML et les schémas représentatifs sont créés à l'aide de « Microsoft Visio 2010 » dans sa version professionnelle, un logiciel développé par Microsoft pour la modélisation, il fait partie de la suite bureautique « Microsoft Office » :

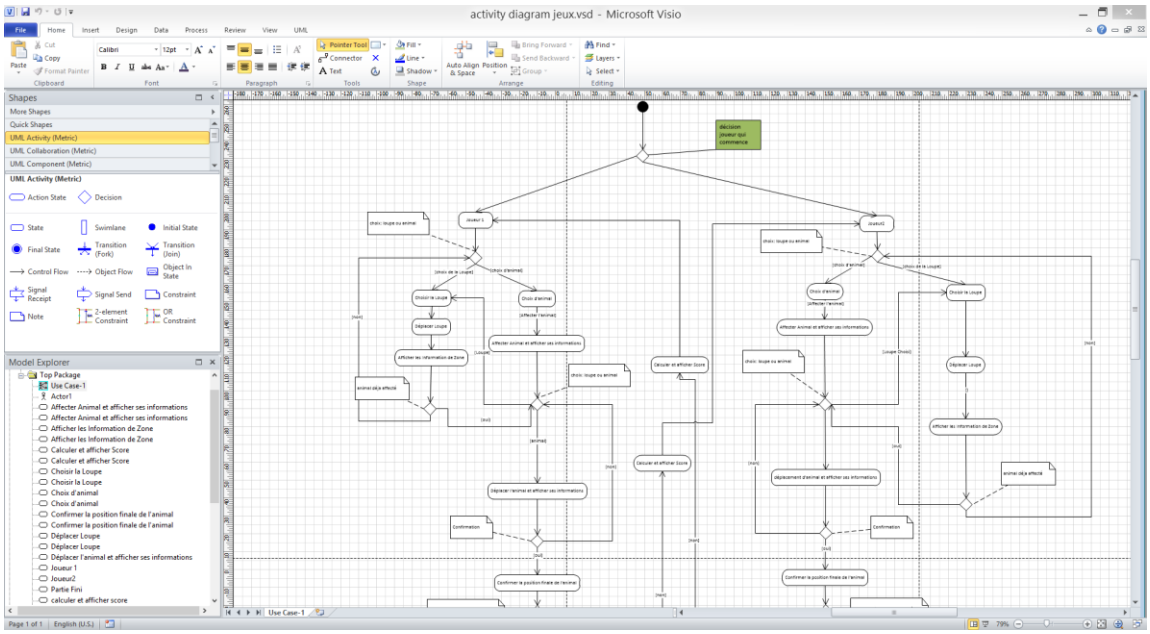


Figure 15 : Microsoft Visio 2012, UML Editor

3) Méthodologie :

Au cours de mon stage, La méthode « Agile » a été utilisée dans toutes les étapes. Cette dernière est une méthodologie pour la modélisation et développement de systèmes informatiques, elle est basée sur les meilleures pratiques. Nous avons choisi cette méthode car elle est plus souple que les méthodes

de modélisation traditionnelle, ce qui en fait un meilleur ajustement dans un environnement qui évolue rapidement. On peut revenir sur un point de développement ou de conception facilement ce qui permet de répondre aux demandes des clients à tout moment.

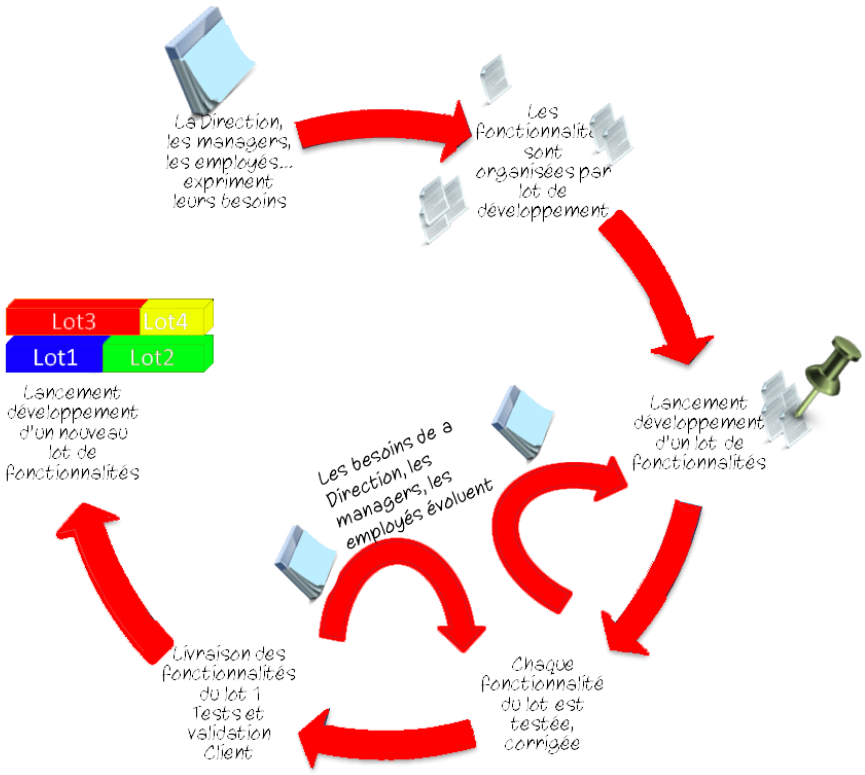


Figure 16 : Méthode Agile

Ma démarche respecte la satisfaction des clients et non les termes contractuels :

1. Découpage de projet sur de petits blocs (selon les fonctionnalités à développer).
2. Hiérarchiser les petits blocs en fonction des besoins (regrouper les fonctionnalités selon leurs besoins).

Cela permet d'éviter le superflu et se concentrer au début de chaque cycle sur ce qui a de la valeur pour le client. L'agilité offre une meilleure visibilité et permet d'éviter les dérives qui arrivent parfois lorsque les développeurs sont isolés.

PARTIE V. Tangible Framework:

1) Etude préalable et conception de la Table Tangible :

a) Un Bref Historique:

RFID est un terme anglais signifie « **R**adio **F**requency **ID**entification» et en français « l'identification par radiofréquence ». C'est une technologie de détection des identificateurs à distance à l'aide de puces électromagnétiques. Un domaine crée après la seconde guerre mondiale, le marché de RFID est en état d'explosion avec plus de 3.98 milliards du tag vendu chaque année et plus de 7.5 milliards de dollars de revenu en 2012. Un système RFID se compose essentiellement au moins de deux composants principaux, un lecteur RFID (RFID puce) et un support de données RFID (RFID tag).

b) Constructeur de la table :

RFIDées une entreprise crée en octobre 2007 à Grenoble France, spécialisée dans le secteur d'activité de l'ingénierie, études techniques basées sur la technologie RFID, cette entreprise prend en charge la construction de la table tangible pour notre projet. Par contre « ColocArts » assurera l'achat et le montage nécessaire du matériel suivant : une table RFID 47 pouces (120cm), les tags RFID nécessaires, un ordinateur ainsi qu'un ventilateur.

c) Tag RFID:

Un tag RFID est le transporteur de données lié à une antenne, qui transmet ses données vers une puce RFID lorsqu'il est placé dans sa zone de détection.

Figure 17 : Tag RFID

d) Puce RFID :

Une puce RFID c'est un capteur RFID. La zone de détection de la puce est une sphère de rayon réglable par le constructeur. La puce détecte la présence d'un tag et elle permet de lire l'information attachée à ce tag détecté.

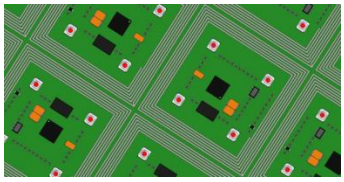


Figure 18 : Puce RFID

e) RFID Slab (dalle en français) :

Une dalle RFID est composée d'un ensemble de puces RFID, qui représente une matrice bidimensionnelle (2D), chaque puce d'une dalle représente une position sur celle-ci. La position d'une puce commence par (1,1) en haut gauche de la dalle (comme montre la figure suivante). Une dalle est liée à une carte réseau unique et spécifique qui sert à transmettre les identificateurs et les positions des tags détectées par la dalle.

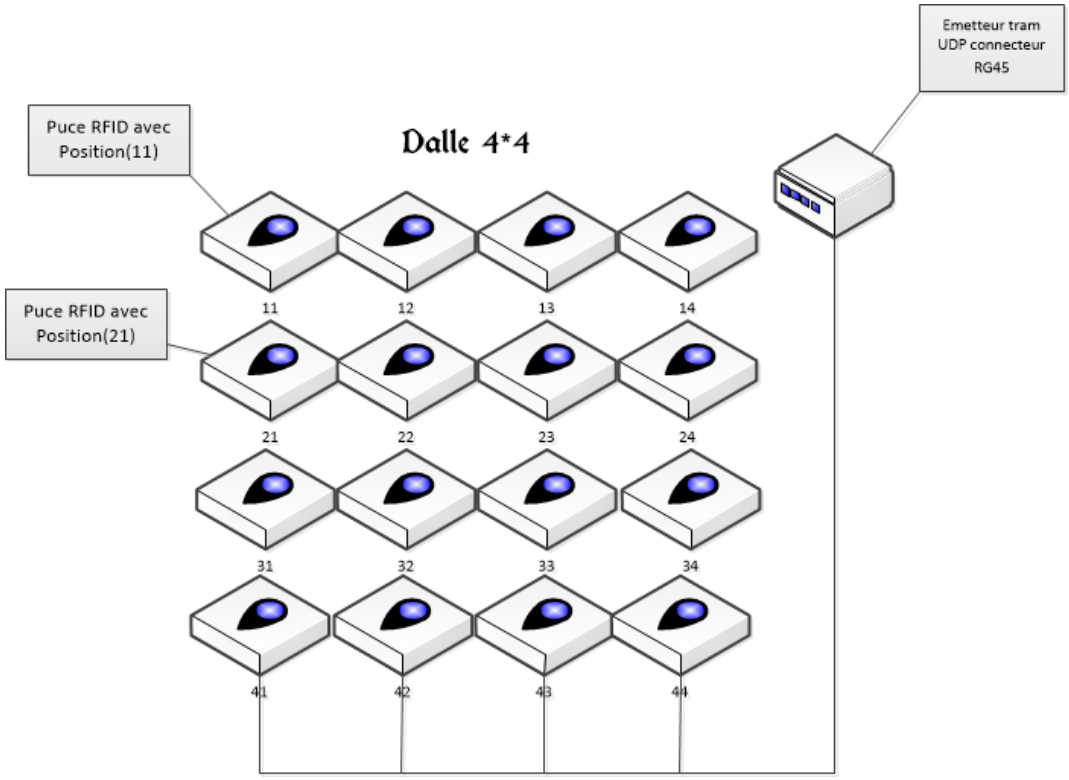


Figure 19 : Architecture Dalle 4X4 Puces

f) Table Tangible :

Contrairement à l'ancienne technologie de la table qui est munie d'un système de diodes lumineuses, la nouvelle version de la table est munie d'un écran LCD HD placé au-dessus des capteurs RFID.

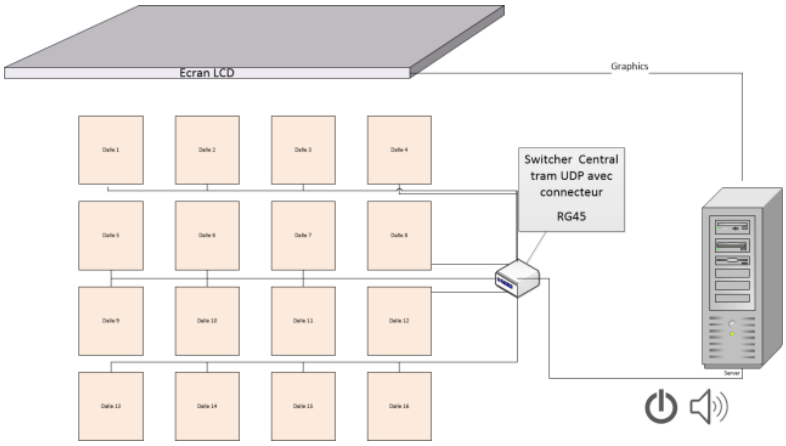


Figure 20 : Architecture de la Table Tangible

g) Trame Tangible de la table:

Tous les octets composants les trames sont au format ASCII. Le format des trames suit le modèle :

[STX]	Commande	:	corps	[ETX]	[horodatage]

Figure 21 : Format de Trame Commande Tangible

S'il s'agit d'une réponse, [ACK] (commande traitée avec succès) ou [SYN] (échec du traitement de la commande) sont ajoutés au début de la trame :

[Ack] ou [SYN]	[STX]	Commande	:	Corps	[ETX]	[horodatage]

Figure 22 : Format de Trame Ack/SYN Tangible

L'horodatage est au format : HHMMSSmmm (heures, minutes, secondes, millisecondes). Les commandes et corps des trames peuvent être amenés à changer. Seul le port 65000 (UDP et TCP) des dalles est ouvert. Voir annexe 6 pour plus des détails sur les différentes commandes.

h) Mode de fonctionnement de la table tangible :

Le mode utilisé avec la table tangible est le mode autonome, la dalle lance elle-même les séquences d'inventaire, et envoie des trames sous le protocole UDP (identificateur des tags présent et leur positions sur la dalle) à l'adresse IP spécifiée par la commande “setserverip”. Même si le mode autonome est activé, la dalle répond quand même aux commandes qui lui sont envoyées.

2) Conception du « Tangible Framework »:

Le fonctionnement global de mon framework est simple : un écouteur qui va faire l’écoute sur le réseau, et s’il détecte une nouvelle trame qui vient de la table, le traitement suivant sera exécuté :

1. Conversation de la trame et extraction des données encapsulées (Frame data et Frame time).
2. Comparaison des données de la trame avec les éléments de « «Frame Buffer », et le traitement suivant s’exécute:
    a. 1er cas (valeur changé) : Mise à jour des positions des tags
        i. Mise à jour du :
            1. Buffer des tags de la vue « Slab ».
            2. Buffer des tags de la vue « Table » en se basant sur:
                1. les valeurs de la position du tag de la vue « Slab ».
                2. Les positions des dalles sur la table (calculé au début automatiques à partir d’un ensemble des paramètres statiques)
        ii. Lissage des positions dans les deux Buffers.
        iii. Déclencher un évènement (modification de la position d’un tag).
    b. 2^{éme} cas (valeur inchangée) : Rien ne se passe.

3. Retour à l'écouteur.

La figure suivante explique l'architecture générale du « Tangible Framework » :

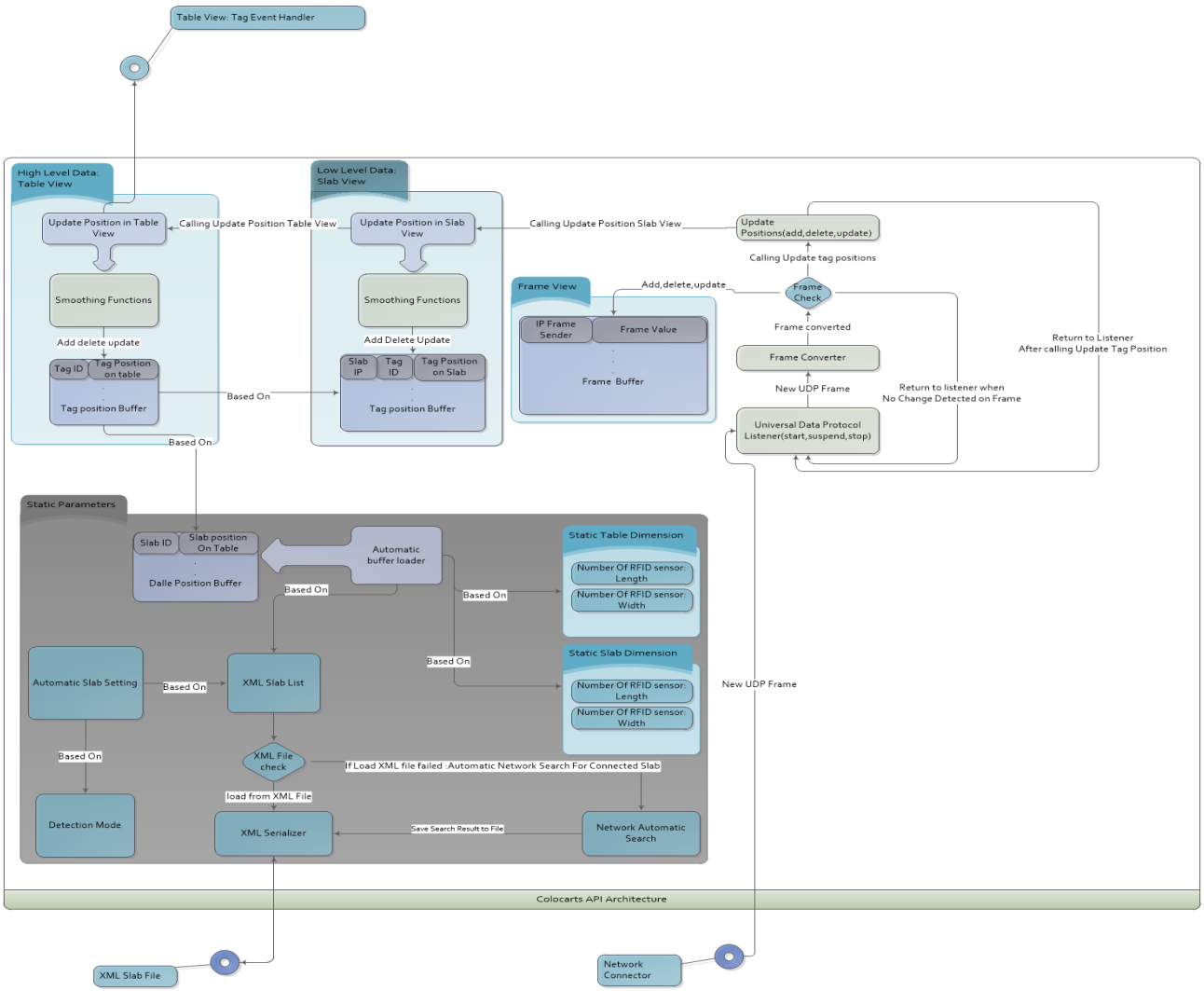


Figure 23 : Architecture de la Framework tangible ColocArts

3) Développement et Documentation du Framework:

a) Développement :

Notre framework est une bibliothèque de liens dynamiques (DLL), cette bibliothèque de classes est générée par « Visual Studio 2013 », écrite en c#, et dans une framework cible « .Net 2.0 » (version de base). Le choix de« .Net 2.0 » est fait pour une future intégration dans Unity3D (Unity ne prend pas en charge au-delà du 2.0), chaque ligne du code est commentée pour faciliter (voir Annexe « Framework Tangible ») :

- La révision du code.
- La mise à jour du framework.
- La documentation du framework.

Une Web-documentation a été créé et publié en se basant sur les commentaires du code et mon framework se base seulement sur les références de base de « .Net 2.0 » :

- **System** : la référence de base de notre Framework (utilisée par toutes les classes).
  - **System.Collections**: fournit des classes spécialisées pour le stockage et l'extraction de données (List, Dictionnary…).
  - **System.IO** : contient des méthodes pour lire et écrire des fichiers ainsi que l'utilisation de Flux (seulement la Classe « TableServer.cs » et la classe « Serialization.cs »).
  - **System.Net** : utilisé pour la programmation des fonctionnalités réseaux (seulement la Classe « TableServer.cs »).
  - **System.Net.NetworkInformation** : utilisé pour la programmation des fonctionnalités réseaux (seulement la Classe « TableServer.cs »).
  - **System.Net.Sockets** : utilisé pour la programmation des fonctionnalités réseaux (seulement la Classe « TableServer.cs »).
  - **System.Text** : utilisé pour codage et décodage des trames réseaux (seulement la Classe « TableServer.cs »).
  - **System.Threading**: utilisé pour permettre d'exécuter la Framework sur un thread indépendant du thread qui l'a créé (seulement la Classe « TableServer.cs »).
  - **system.XML :** utilisé seulement pour le but de la sérialisation et la désérialisation (seulement classe Serialization.cs).

b) Documentation :

Après la phase de développement, j'ai Construis une Web-Documentation en anglais (car l'anglais est le plus utilisé) à l'aide du Plugin gratuit de Visual Studio « Sand Castle » qui produit une documentation précise et complète en se basant sur l'ensemble du code source et ces commentaires (chaque commentaire suit la forme XML suivante) :

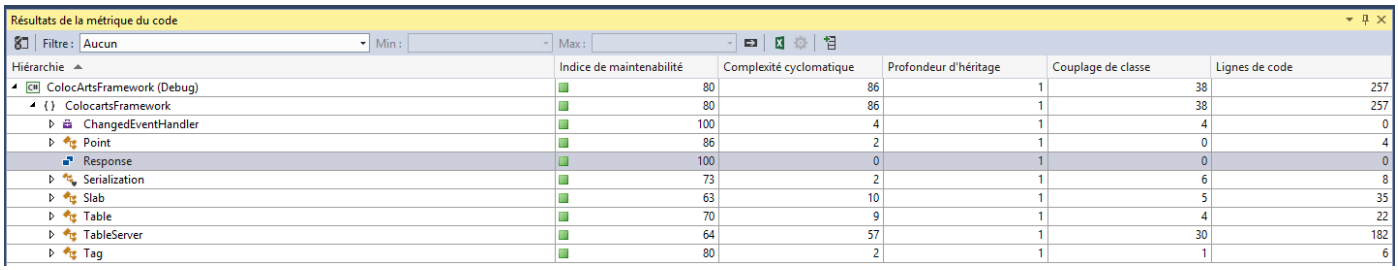
```
        ///<summary>
/// Description de la fonction/ methode de la classe/ membre de la classe/ classe
///<param name="paramètre 1">description du paramètre 1</param>
///<return> description de type de retour si une fonction</return>
///<example>exemple d'utilisation</example>
///</summary>
```

Cette Web-Documentation a été publié ici.

c) Analyse de code du « Tangible Framework »:

Microsoft Visual Studio nous offre la possibilité d'analyser avec un jeu de mesures notre code, il permet aux développeurs d'identifier les futurs risques potentiels dans la maintenance où la mise à jour du code et il donne une visibilité plus clair sur l'état actuel du projet :

- **Indice maintenabilité** : mesure de la facilité de la maintenance du code, les valeurs hautes sont préférables.
- **Complexité cyclomatique** : mesure de nombre de branches, les valeurs basses sont préférables.
- **Profondeur d'héritage** : mesure la longueur de la hiérarchie d'héritage d'objet, les valeurs basses sont préférables.
- **Couplage de classe** : mesure le nombre de classes référencées, les valeurs basses sont préférables.
- **Lignes de code** : se rapprochent de lignes de codes exécutables, les valeurs basses sont préférables.


Résultats de la métrique du code

Filtre : Aucun Min : Max :

Hiérarchie	Indice de maintenabilité	Complexité cyclomatique	Profondeur d'héritage	Couplage de classe	Lignes de code
ColocArtsFramework (Debug)	80	86	1	38	257
ColocartsFramework	80	86	1	38	257
ChangedEventHandler	100	4	1	4	0
Point	86	2	1	0	4
Response	100	0	1	0	0
Serialization	73	2	1	6	8
Slab	63	10	1	5	35
Table	70	9	1	4	22
TableServer	64	57	1	30	182
Tag	80	2	1	1	6

Figure 24 : Analyse du Code de la « Tangible Framework »

PARTIE VI. La Biodiversité au bout des doigts version 2

1) « La Biodiversité au bout des doigts »version 2 :

a) Etude préalable :

Le jeu demandé diffère de celle de « Biodiversité au bout des doigts » car le client ne veut pas d'écran supplémentaire pour afficher les informations pratiques. Nous allons donc, en partenariat avec RFIDées, concevoir une table répondant à ses attentes.

La dite table aura un « plateau de jeu RFID » plus petit que sur le modèle de référence, afin d'y intégrer une place pour afficher les informations pratiques en question. En voici un exemple :

Figure 25 : nouvelle technologie Table RFID

La largeur de la partie « écran infos » est de 15cm. Dès lors, la diagonale de l'écran de jeu est de 93.6cm (73.5cm de large x 58cm de profondeur).

La carte du jeu est fournie par la Mairie de Gresse En Vercors, au format HD (720p minimum), afin que cette dernière puisse être intégrée au jeu. La carte définitive concernera la zone ci-dessous :

Figure 26 : Carte Gresse En Vercors

b) Les animaux :

La liste des animaux est la suivante : Vautour fauve, Loup, Aigle Royal, Renard, Marmotte, Tétras Lyre, Bouquetin, Chamois, Hérisson et Papillon.

Les animaux disposent de fiches illustrées de présentation dédiées afin d'être présentés aux joueurs avant de jouer leur tours (sur la partie de l'écran RFID dédié « écran infos »). Ces fiches seront fournies par la mairie de Gresse En Vercors.

Pour chaque animal, la marie de Gresse En Vercors fournira une figurine représentative à ColocArts qui s'occupera d'intégrer un tag RFID dans chaque figurine pour que ces dernières soient reconnues par la table.

c) Les milieux :

La liste des milieux est la suivante : le village, espace cultivé, hêtraie-sapinière, pelouse montagnarde, éboulis, falaise, forêt de pin à crochet, pelouse d'altitude.

Ces milieux disposent de fiches de présentation qui apparaissent afin d'être présentées aux joueurs (sur la partie de l'écran RFID dédié "écran d'info"). Le contenu de ces fiches sera fourni par la mairie de « Gresse en Vercors ».

Ces présentations apparaitront quand le joueur passera un objet type loupe (fournies par la mairie de « Gresse-en-Vercors ») sur la carte de jeu.

d) Interactions Entre Animaux:

Les interactions entres les animaux préalablement cités sont les suivantes :

	1 – Vautour fauve	2 -Loup	3 – Aigle royal	4 -Renard	5 -marmotte	6 – Tétras Lyre	7 – Bouquetin	8 – chamois	9– hérisson	10 – Papillon
1 – Vautour fauve		1	1	1	1	1	1	1		
2 -Loup					1	1	1	1		
3 – Aigle royal					1	1	1	1		
4-Renard					1				1	1
5 -Marmotte										
6 – Tétras Lyre										1
7 – Bouquetin										
8– chamois										
9 – hérisson										1
10 – papillon										

se lit en ligne — 1Mange — peut être mangépas souvent

Figure 27 : Tableau Interaction Animaux

La liste verticale « mangeant » la liste horizontale, on comprendra pour la première ligne que le Gypaète mange le sanglier, et non l'inverse. Au même titre, pour la neuvième ligne, la mésange mange la fourmi.

e) Interactions Avec Zones géographique:

Les interactions, du point de vue géographique, seront disposées ainsi qu'il suit :

	A – Village	B - E space cultivés	C – Hétraies-sapinière	D – Pelouse montagnarde	E – Eboulis	F – Falaise	G – Forêt de pin à crochet	H - Pelouse d'altitude
1 – Vautour fauve	0%	0%	0%	100%	100%	100%	50%	100%
2 -Loup	0%	50%	100%	100%	50%	0%	100%	50%
3 – Aigle royal	0%	50%	0%	100%	100%	100%	50%	100%
4 – Renard	50%	100%	100%	100%	50%	0%	50%	0%
5 -marmotte	0%	0%	0%	100%	100%	50%	0%	100%
6 – Tétras Lyre	0%	0%	0%	100%	0%	0%	100%	50%
7 – Bouquetin	0%	0%	0%	100%	100%	50%	50%	100%
8 – Chamois	0%	0%	50%	100%	100%	50%	50%	100%
9 – Hérisson	100%	100%	50%	0%	0%	0%	0%	0%
10 – papillon SL	100%	100%	50%	100%	100%	50%	50%	100%

100% niche et chasse
50% de passage
0% rarement

Figure 28 : Tableau Interaction Animal/Zone

On comprendra alors que la marmotte se trouve très souvent, voir majoritairement dans les zones « pelouse de montagne, éboulis et pelouse d'altitude » pour y chasser et y vivre, et dans une moindre mesure dans la « forêt de pin ». Néanmoins, rien ne l'empêche d'aller dans les autres milieux.

Les animaux ont donc un pourcentage d'interaction différent selon leur espèce et le lieu où ils seront placés sur la carte. Voici la carte représentant les différentes zones :

Figure 29: Zones Géographiques de Vercors

De plus, chaque milieu pourra être présenté si l'utilisateur place un tag RFID sur la carte. L'écran d'information affichera alors, comme pour les fiches des animaux, des informations liées au milieu en question. Les informations sur les lieux seront fournies par la mairie de « Gresse En Vercors ».

Quand le joueur placera un animal sur la carte, cette dernière reconnaitra immédiatement l'animal dont il est question. Si l'animal n'est pas à la meilleure place, cela impactera le score biodiversité. A l'inverse, si l'animal est « bien placé » le score biodiversité du joueur augmentera.

f) Le scénario global

Deux joueurs jouent sur la table RFID. Le jeu se déroule « chacun son tour ». Avant de jouer, les deux joueurs doivent prendre 5 petites figurines chacun. Ces figurines représentent les animaux du jeu. Les figurines sont équipées de tags RFID afin que l'écran puisse « savoir » où sont les animaux, qui sont-ils, à quel joueur ils appartiennent (joueur 1 ou joueur 2 représentés par des couleurs différentes) et quelles sont les interactions possibles.

Avant chaque tour, le joueur qui s'apprête à jouer, pose l'animal qu'il a choisi sur un point précis de la table afin que des informations sur ce dernier lui soient communiquées (les informations sont celles des fiches de présentation des animaux).

Une fois les informations lues, le joueur pose alors son animal ou bon lui semble sur la carte de jeu interactive. Le second joueur choisit alors un animal à son tour, le pose sur la partie dédiée, reçoit les informations et place son animal sur la carte interactive également.

Au fur et à mesure que le jeu avance, les joueurs voient les interactions se produire. On imagine que si un joueur place l'aigle près d'une marmotte, l'interaction sera alors « l'aigle mange la marmotte » et sera représenté visuellement sur la table.

Comme vu en amont dans cette partie, les interactions sont entre les animaux mais également dépendantes du milieu ou les animaux sont placés. Les interactions entrainent des points qui permettent de « noter » la connaissance de la biodiversité des joueurs. Plus les joueurs mettent les animaux « au bon endroit » plus le score de biodiversité sera élevé. A l'inverse, si le joueur place les animaux à des endroits « non pertinents », le score de biodiversité diminuera. La bonne position des animaux entraine donc un score en biodiversité plus élevé que si les joueurs mettent les animaux à des endroits non pertinents.

Ainsi, si l'on reprend notre exemple de l'aigle et de la marmotte, celui-ci peut permettre d'obtenir plusieurs scores différents en fonction de la ou sont placés les animaux. Si les deux animaux sont placés dans le milieu « pelouse montagnarde » par exemple, le score sera donc maximum pour cette interaction pour deux raisons :

1. Les deux animaux ont 100% d'interaction dans ce milieu.
2. L'aigle mange la marmotte.

Les joueurs posent donc tours à tours les animaux sur l'écran dans le but d'obtenir un score « biodiversité » maximum en essayant de placer les bons animaux aux bons endroits.

Une fois tous les animaux posés, la partie est finie. On passe alors un objet de type « brosse à tableau » sur la table pour effacer la partie finie et en recommencer une nouvelle.

2) Conception du jeu biodiversité version 2 :

a) Diagramme d'activité globale du jeu :

Le diagramme suivant montre les différents états de déroulement d'une partie de jeu entre deux joueurs :

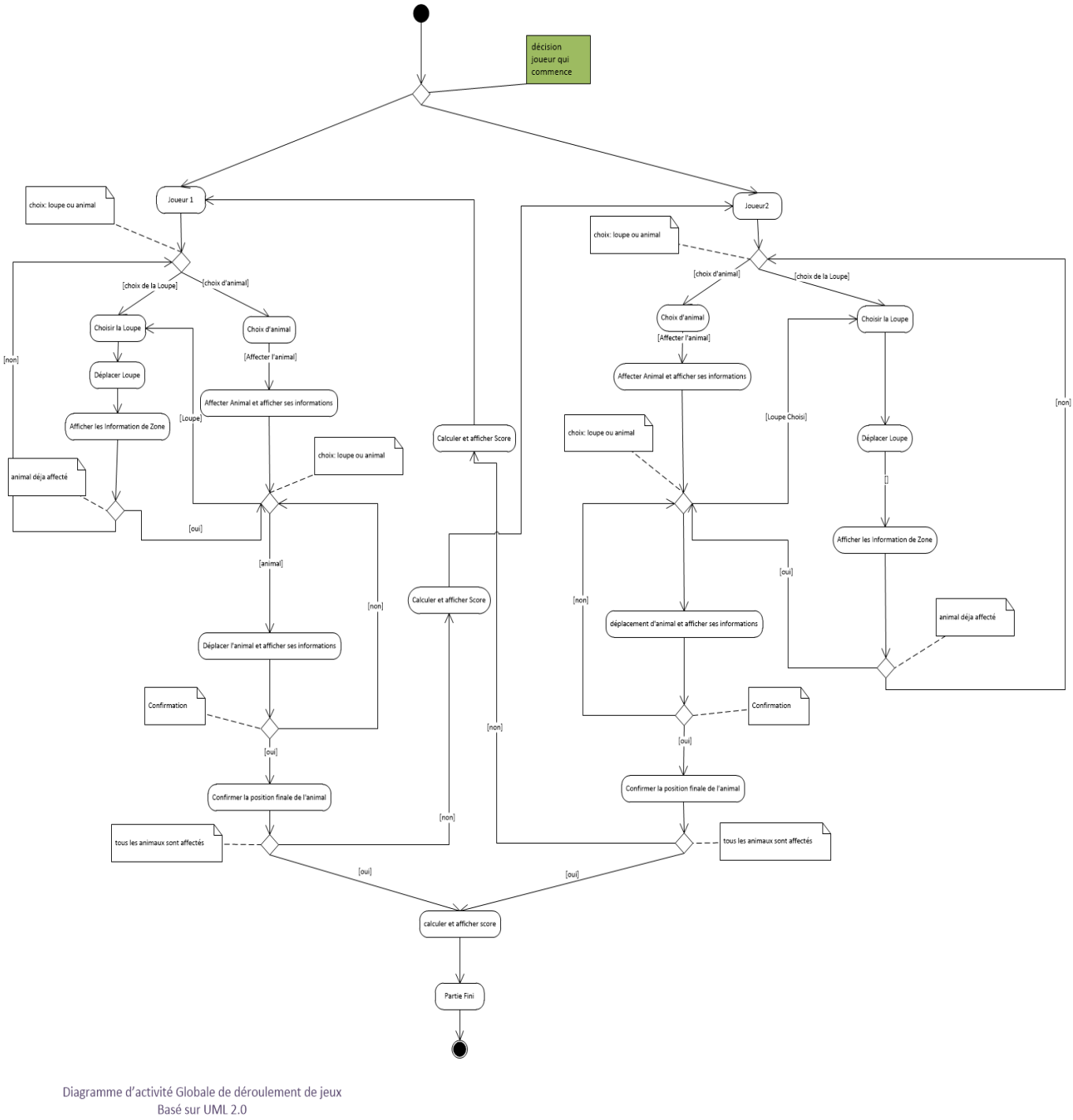


Diagramme d'activité Globale de déroulement de jeux
Basé sur UML 2.0

Figure 30 : Diagramme d'activité Globale de jeux

b) Diagramme de séquence de déplacement d'un tag dans le jeu :

La figure suivante montre le diagramme de séquence des appels aux fonctions lors d'un déplacement d'un objet tangible sur la table, et la réaction du chaque intervenant de notre jeu :

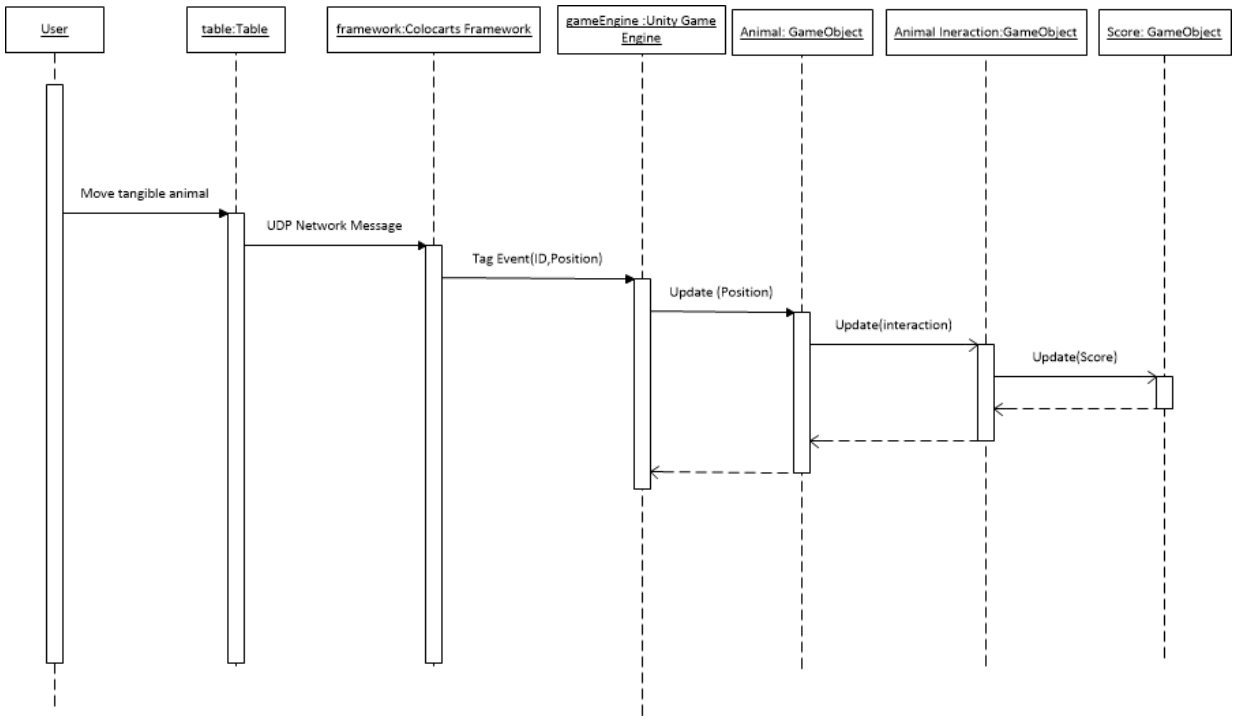


Figure 31 : diagramme de séquence, Mouvement d'un animal dans le jeu Biodiversité

3) Réalisation du jeu :

Le jeu est développé avec Unity 3D version 4.5 professionnel, programmation c# (choix argumenté précédemment), nos buts :

- Trouver le Gameplay adéquate au scénario demandé par la marie de « Gresse de Vercors ».
- Pousser le rendu du graphique au maximum.
- rendre l'expérience de jeu époustouflante.
- Examiner toutes les possibilités de jeu avec la technologie tangible (fixer tous les bugs).
- Gérer toutes les exceptions (matériel et software) pour une exécution du jeu sans échec.
- Trouver la solution adéquate pour la gestion des objets du jeu (animal/zone/loupe/gomme).

Pour plus de détails techniques de développement voir Annexe « Biodiversité aux bouts des doigts version 2».

PARTIE VII. Gestionnaire d'applications pour Table :

1) Conception de la Gestionnaire d'applications:

a) Présentation d'application :

Une application qui gère l'installation, la désinstallation et la mise à jour des applications de la table:

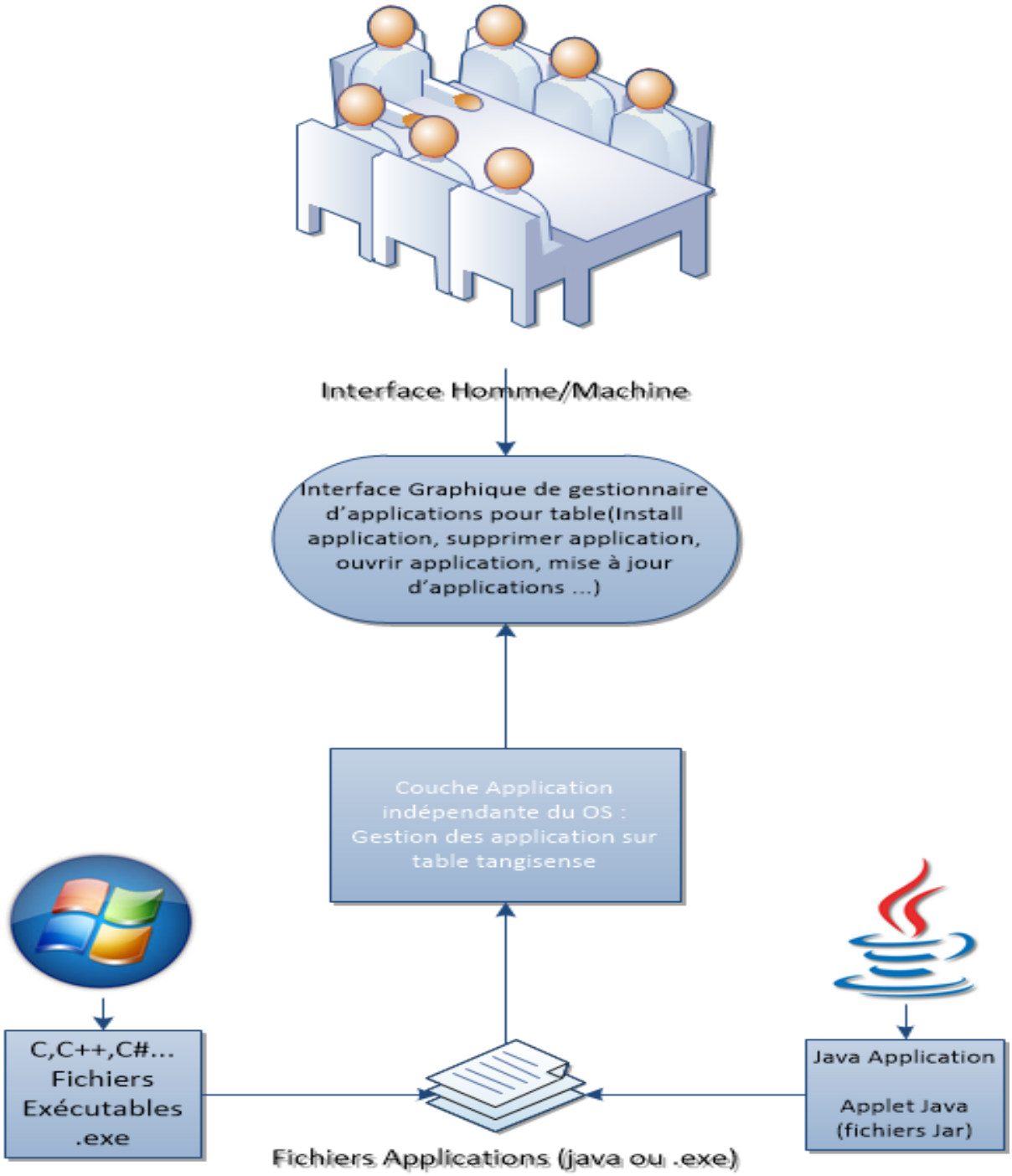


Figure 32 : Architecture de la Gestionnaire d'applications pour la Table tangible

b) Diagramme de cas-utilisations :

Le diagramme suivant décrit toutes les interactions entre l'utilisateur et l'application « Gestionnaire d'applications » :

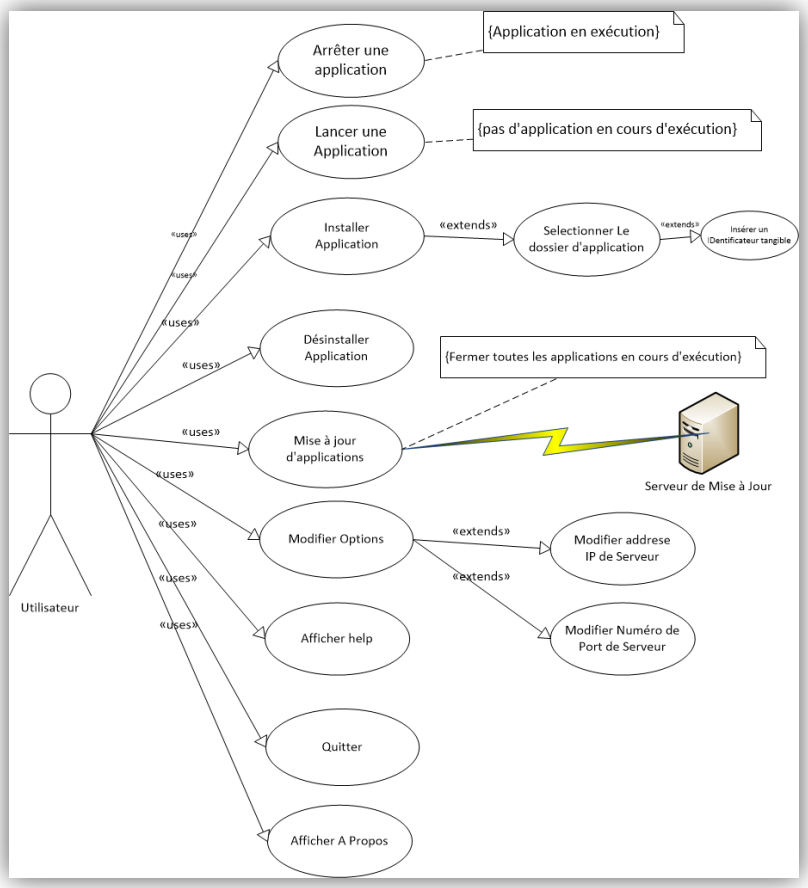


Figure 33 : Diagramme de cas-utilisations Gestionnaire d'applications

c) Diagramme de Séquence mise à jour d'une application:

Le diagramme suivant montre un scénario dans l'ordre chronologique de mise à jour abouti avec succès :

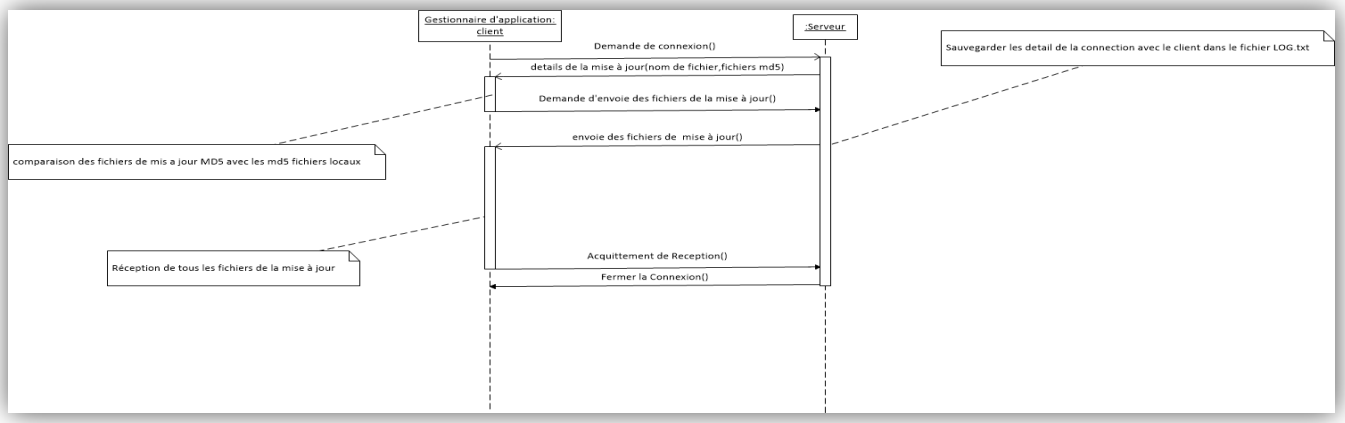


Figure 34 : Gestionnaire d'applications, Diagramme de Séquence Mise à jour d'applications

2) Réalisation et Aperçu d'interface:

La gestionnaire des applications pour la table tangible est développé avec Visual Studio 2013 version professionnel, dans une framework cible « .Net 4.5 », version plus riche que les anciennes versions du « .Net ». Cette application est écrite en c# (langage utilisé pour le développement au sein de l'entreprise, son choix est argumenté précédemment).

Nos buts lors du développement de cette application :

- Un graphique élégant.
- Une facilité dans l'utilisation, même par des non-professionnels.
- Gestion de toutes les exceptions.
- Garantir une belle organisation de toutes les applications de la table tangible.
- Exécution de chaque application (par un objet tangible ou tout simplement par la souris manuellement).
- Garantir la mise à jour de chaque application de la table individuellement.

Pour un aperçu sur les interfaces développées, voir l'annexe « Gestionnaire d'applications ».

PARTIE VIII. Application Serveur de la Mise à jour :

1) Conception du Serveur:

a) Architecture de serveur :

L'architecture de notre application serveur respecte le schéma suivant et elle est sous la forme (1 serveur et n client), l'application serveur accepte n'importe quel demande communication des clients qui veulent établir une connexion avec elle (seulement si le nombre maximale des connexions est atteint : surcharge de serveur) :

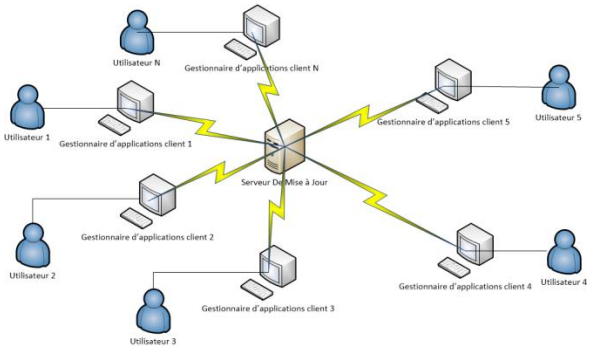


Figure 35 : Architecture de Serveur de mise à jour 1-N

b) Diagramme de cas-utilisations :

La figure suivante montre les différentes interactions entre l'utilisateur et l'application serveur :

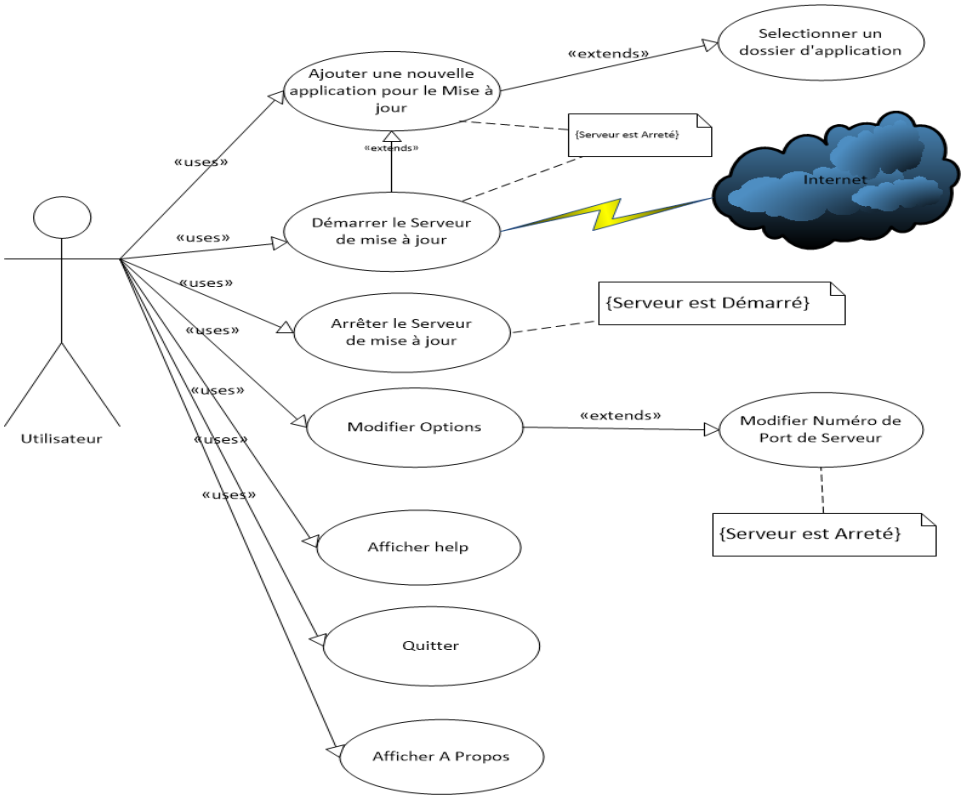


Figure 36 : Diagramme de cas-utilisations Serveur de Mise à Jour

c) Diagramme de Séquence d'un Mise à jour avec succès :

Le diagramme suivant montre un scénario dans l'ordre chronologique de mise à jour abouti avec succès :

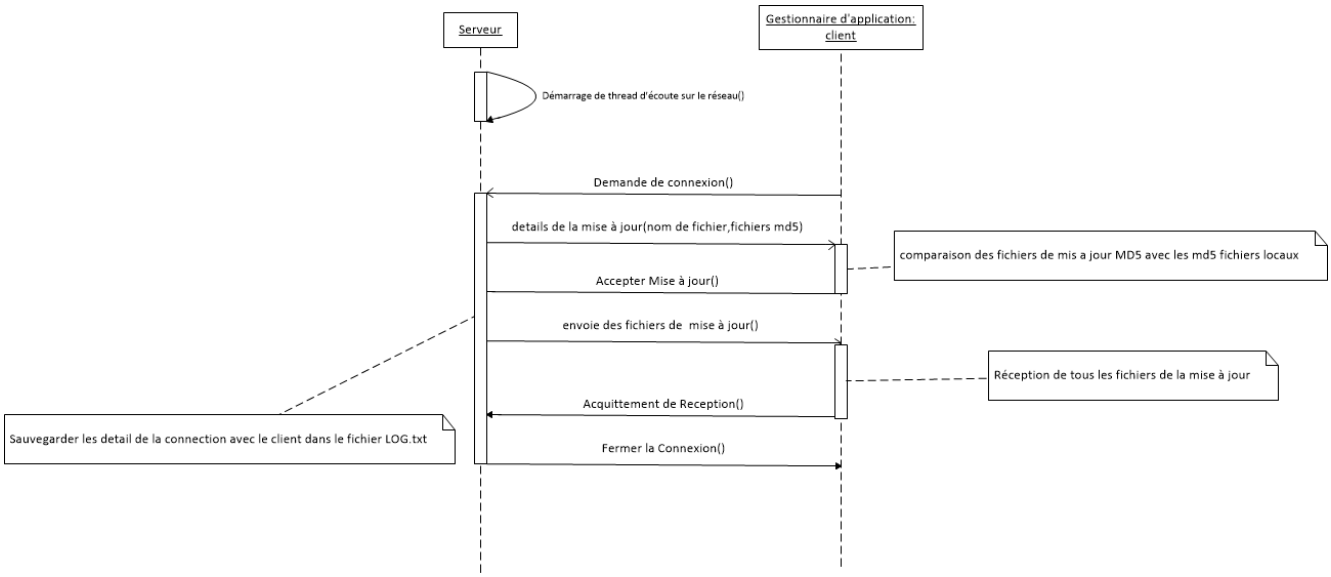


Figure 37 : Serveur de mise à jour, Diagramme de Séquence Mise à jour

2) Réalisation et Aperçu d'interface:

Le serveur de la mise à jour pour la table tangible est développé avec Visual Studio 2013 version professionnel, dans une framework cible « .Net 4.5 », version plus riche que les anciennes versions du « .Net ». Cette application est écrite en c# (langage utilisé pour le développement au sein de l'entreprise, son choix est argumenté précédemment).

Nos buts lors du développement de cette application :

- Un graphique élégant.
- Une facilité dans l'utilisation, même par des non-professionnels.
- Intégration de la possibilité du glisser-déposer les dossiers d'applications pour la mise à jour.
- Gestion de toutes les exceptions.
- Garantir une belle organisation de tous les composants de l'application.
- Rajouter une ou plusieurs applications pour la diffusion (Mise à jour) (utiliser l'interface de rajout ou simplement glisser-déposer les dossiers).
- Garantir un affichage de l'état d'une mise à jour (barre de progression, fichier LOG).
- Un mécanisme de mise à jour adéquate à nos besoins.
- Offrir un service durable et maintenable à long terme.

Pour un aperçu sur les interfaces développées, voir l'annexe « Serveur de Mise à jour ».

PARTIE IX. Perspectives

La technologie tangible est complémentaire à la technologie tactile. Le tactile nous donne plus de précisions mais le tangible nous donne plus d'informations sur l'objet détecté. Dans les futurs projets nous visons à mélanger les deux technologies, pour exploiter au mieux la précision du tactile et la reconnaissance précise du tangible.

Plusieurs futurs projets que nous visons à faire :

- Une application de travail collaborative qui relie plusieurs tables tangible à distance (on peut mélanger les deux technologies) pour discuter, planifier, marquer des notes, visualiser les idées au sein un groupe du travail, entreprise…etc.
- Trouver d'autres idées de jeux pour la table.
- Une application de gestion de planning interactive.

Bibliographie et Webographie

- Encyclopédie du Wikipédia en Français et en Anglais :

  http://fr.wikipedia.org/

  http://en.wikipedia.org/

- Documentation officielle d'Unity 3D :

  answers.unity3d.com

  http://unity3d.com/community

- Documentation officielle de « Visual Studio » et Microsoft c#:

  http://msdn.microsoft.com/en-US/

- Documentation officielle de Java:

  http://docs.oracle.com/javase/7/docs/api/

- Documentation officielle d'Adobe:

  https://helpx.adobe.com/photoshop/topics.html

  https://www.adobe.com/support/documentation/en/illustrator/

- Documentation officielle d'Autodesk « 3D max » :

  http://docs.autodesk.com/3DSMAX/16/ENU/3ds-Max-Help/

- Générateur de la documentation C# Sand Castle :

  http://sandcastle.codeplex.com/

- « Projet de La Biodiversité au bout des doigts » :

  http://multicom.imag.fr/multicom.imag.fr/spip20d9.html?article167

- La bibliothèque des statistiques:

  http://www.planetoscope.com/

- Documentation de « Tangible Framework » :

  http://www.colocarts.com/CAFH/TangibleFramework.htm

Annexe 1 Gestionnaire d'applications

La figure suivante montre l'interface principale de Gestionnaire d'applications :

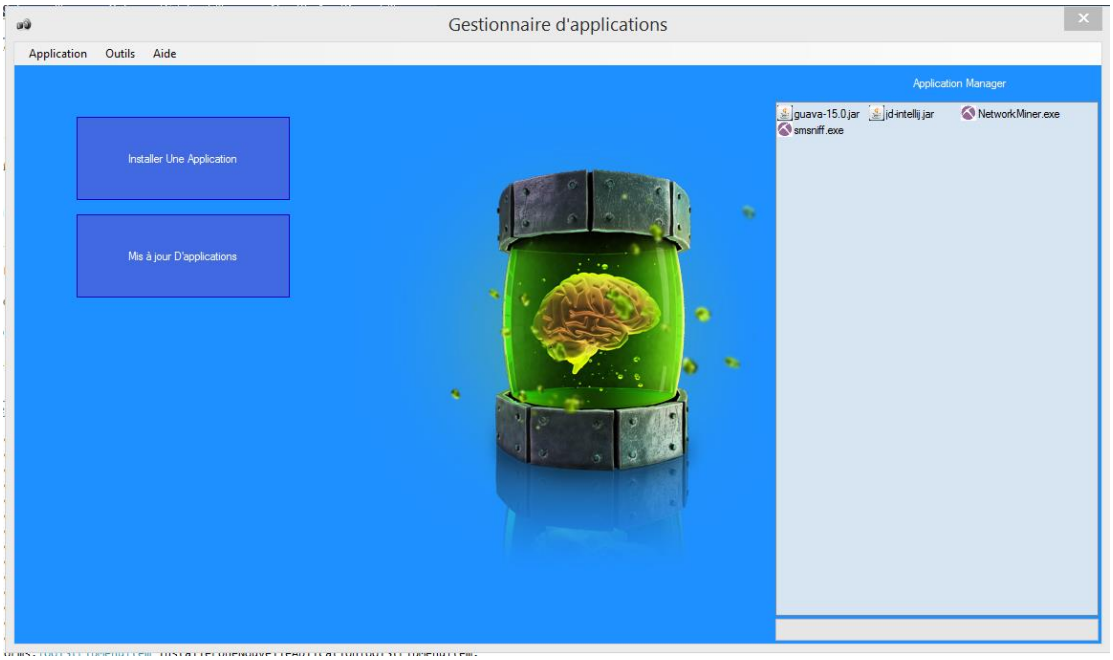


Figure 38 : interface principale de la Gestionnaire d'applications pour la table

La figure suivante montre le menu de contexte qui comporte (Lancer application, Supprimer application, Mise à jour d'application) :

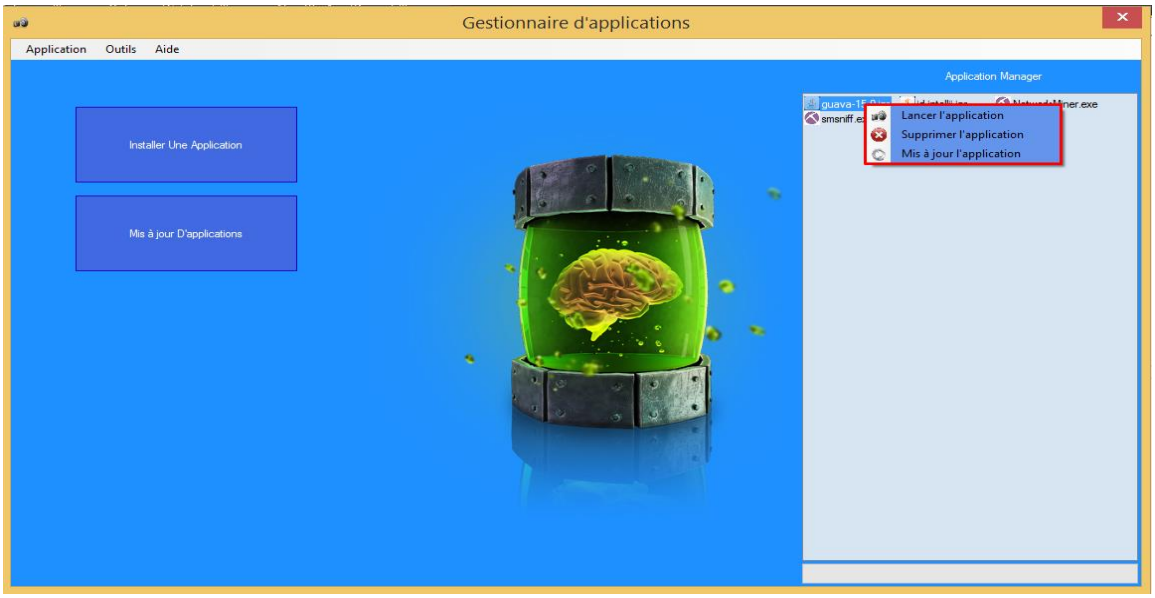


Figure 39 : Menu de contexte de Gestionnaire d'applications

La figure suivante montre l'interface d'installation d'une nouvelle application, une sélection de dossier qui contient les fichiers d'installation :

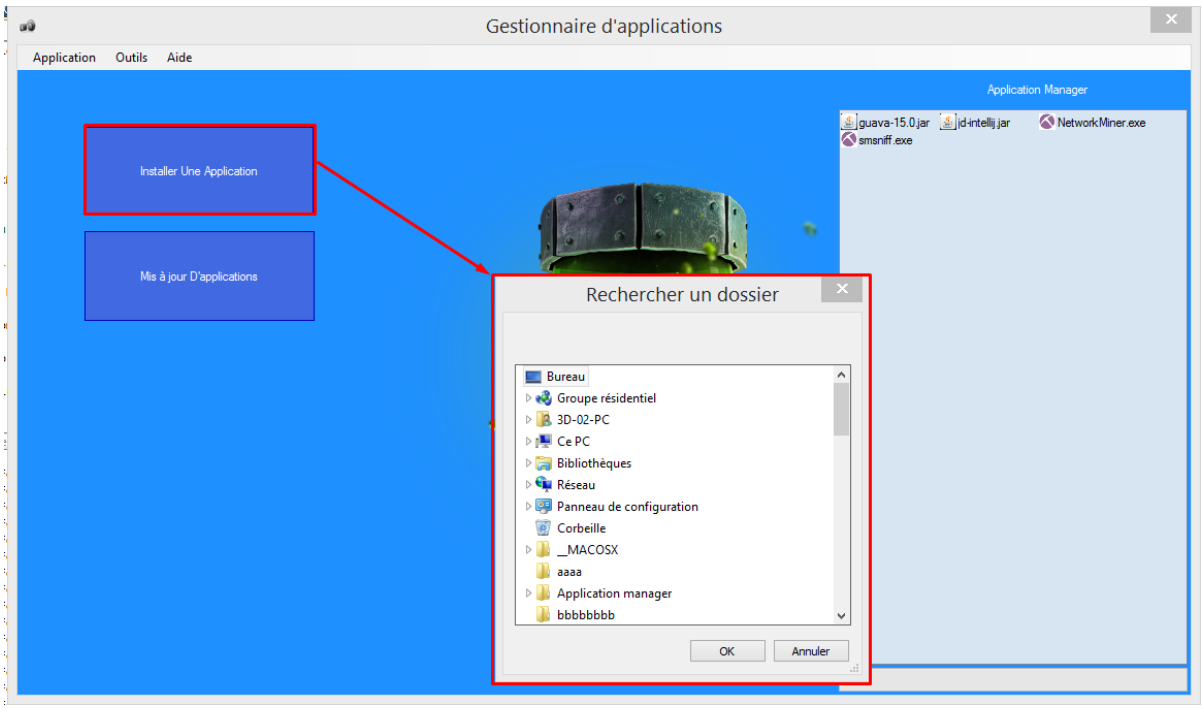


Figure 40 : Installer une application, Gestionnaire d'applications

La figure suivante montre le menu Application (installer une nouvelle Application, Lancer l'application, Supprimer l'application, Mise à jour d'application et Quitter) :

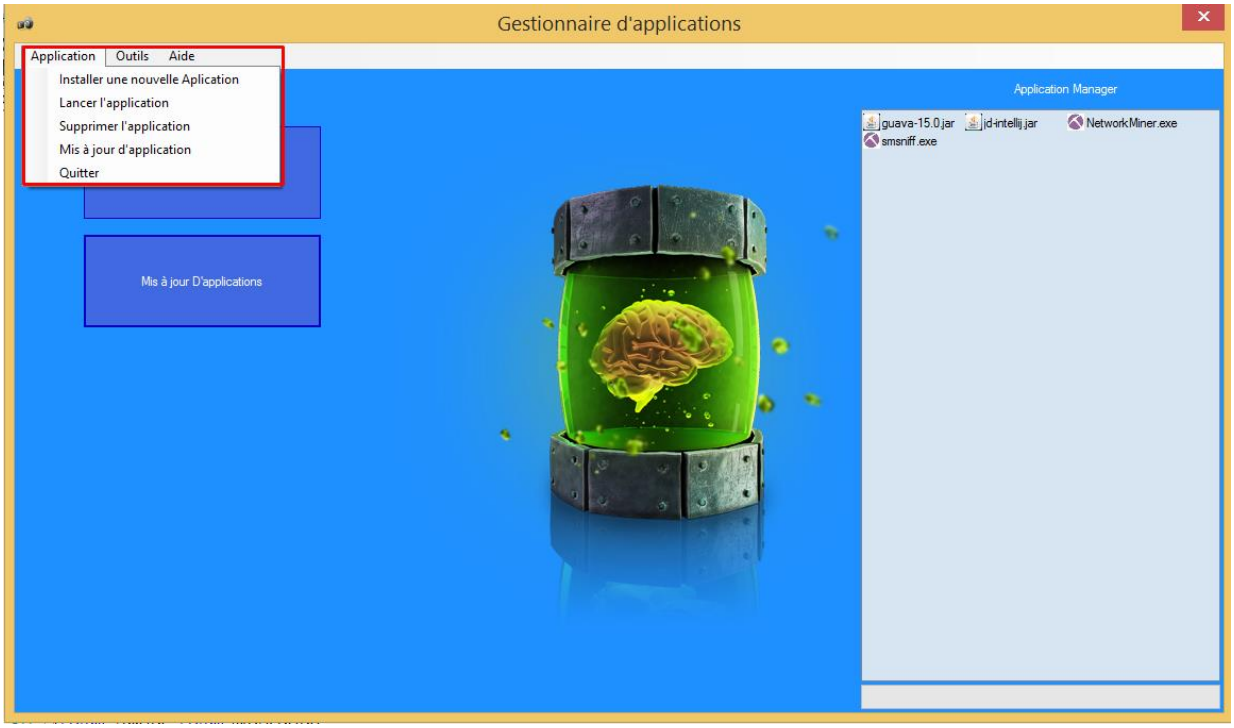


Figure 41 : Menu 1 principale, Gestionnaire d'applications

La figure suivante montre le menu « Outils » qui contient (Afficher et Options) :

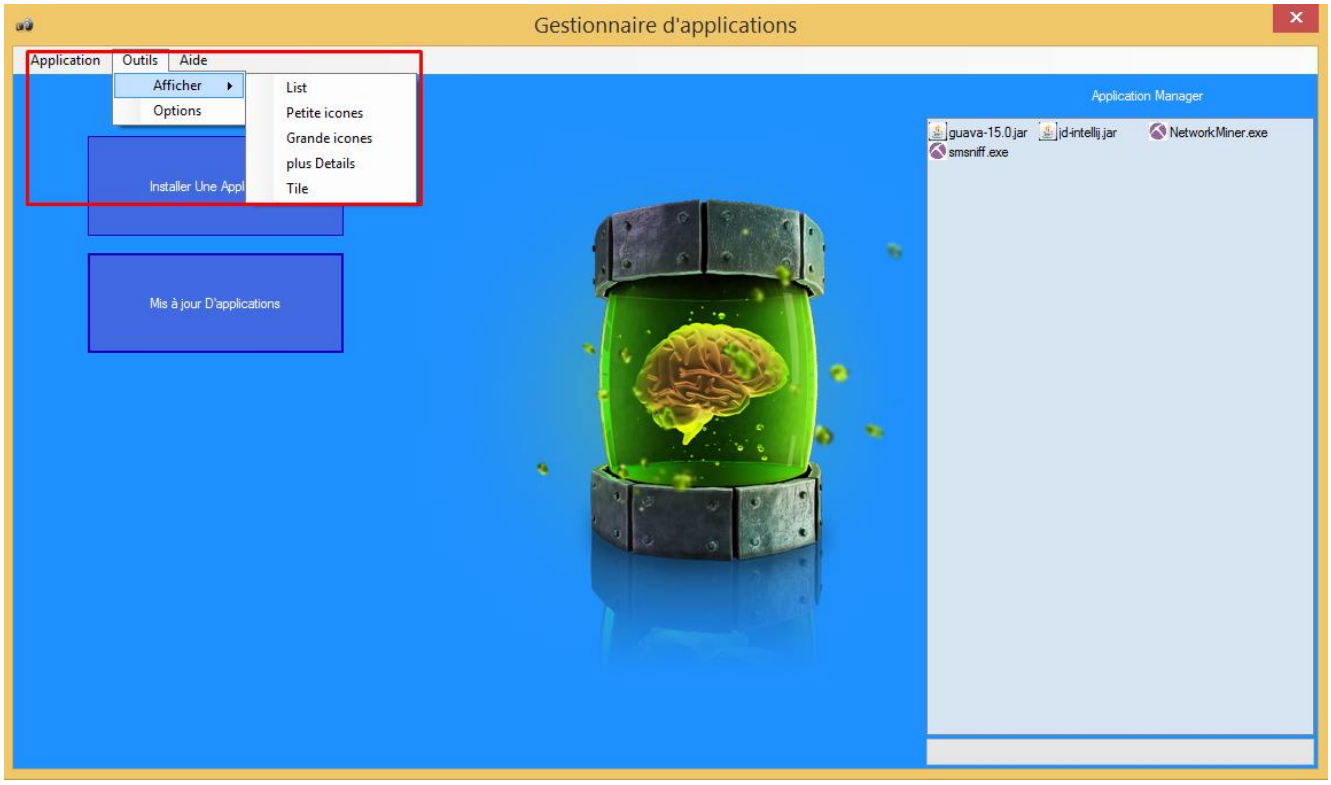


Figure 42 : Menu 2 principale, Gestionnaire d'applications

La figure suivante montre le menu « Help » qui contient (Help et A propos) :

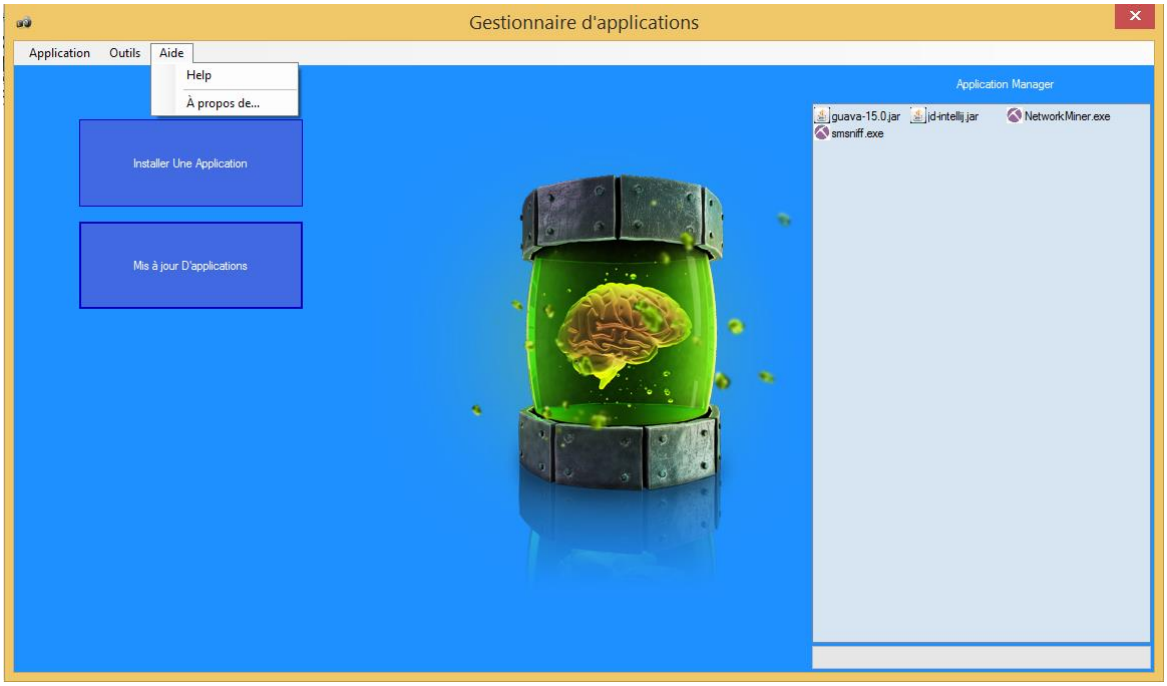


Figure 43 : Menu 3 principale, Gestionnaire d'applications

La figure suivante montre l'interface « Options », à l'aide de cette interface l'utilisateur peut modifier l'adresse IP du serveur de mise à jour ou le numéro de port et sélectionner la fréquence de mise à jour automatique :

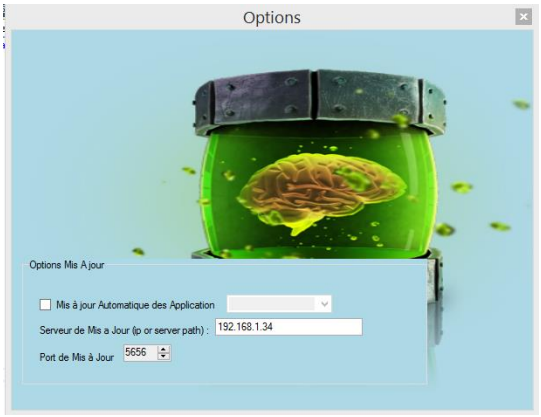


Figure 44 : interface Options pour la Gestionnaire d'applications

Une interface « à propos » qui affiche les détails de l'application (Nom, version, Entreprise de développement, coordonnées avec une petite description) :

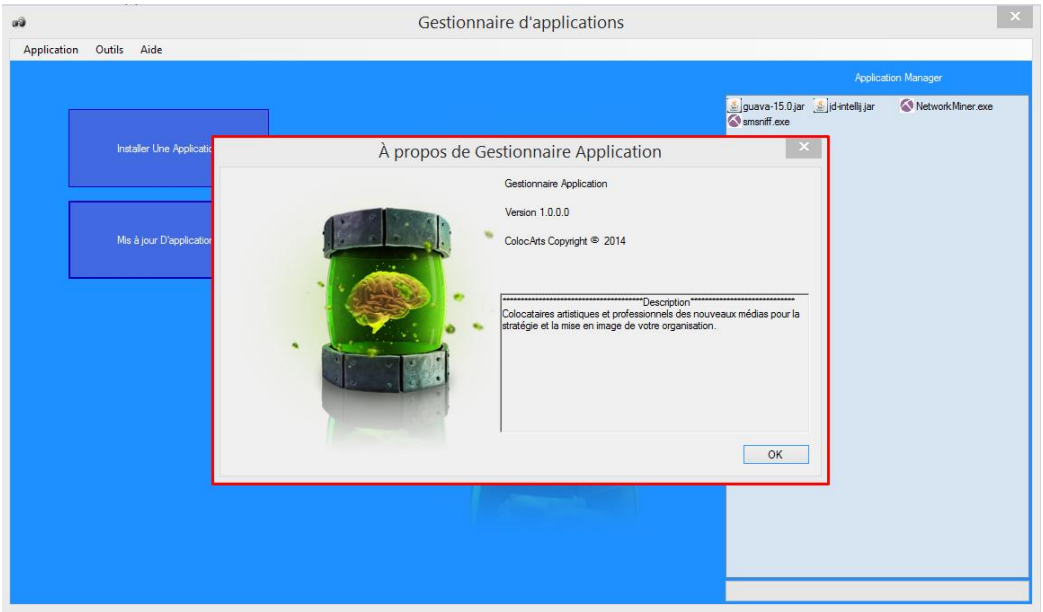


Figure 45 : Gestionnaire d'applications s, interface A propos

Toutes les exceptions et les erreurs qui peuvent être présentes sont gérées dans l'application et voilà un exemple de message d'erreur : en cas d'échec de mise à jour d'une application un message d'erreur s'affiche :

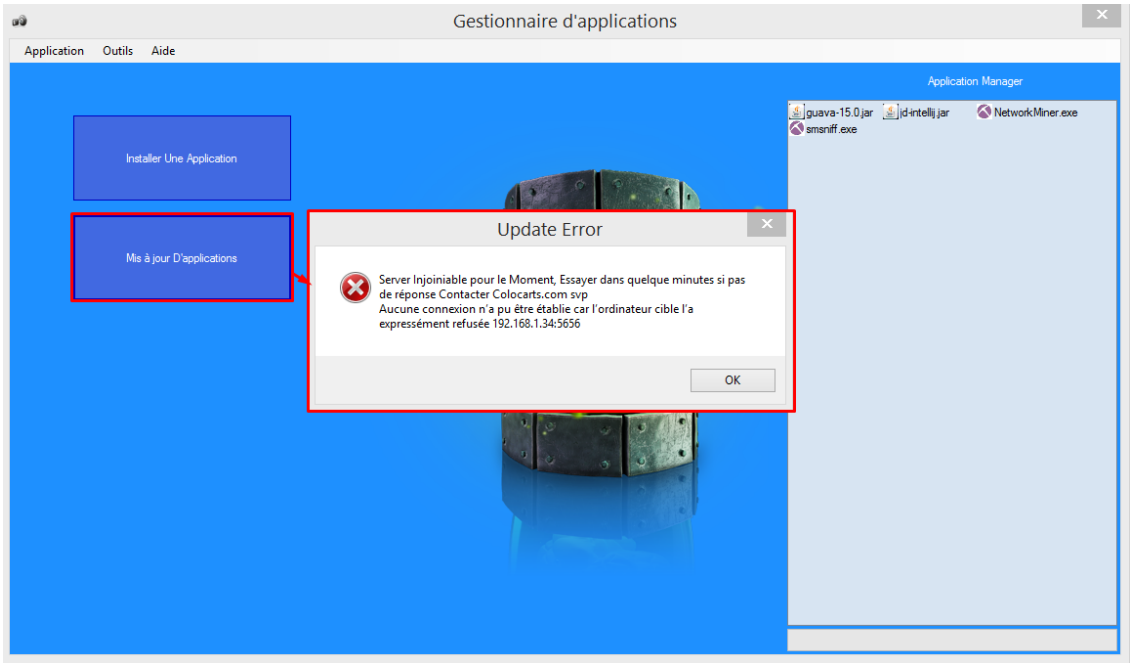


Figure 46 : Gestionnaire d'applications, message d'erreur de mise à jour

La figure suivante montre le processus de la mise à jour en cours (barre de progression et affichage textuel de l'état de la mise à jour en temps réel) :

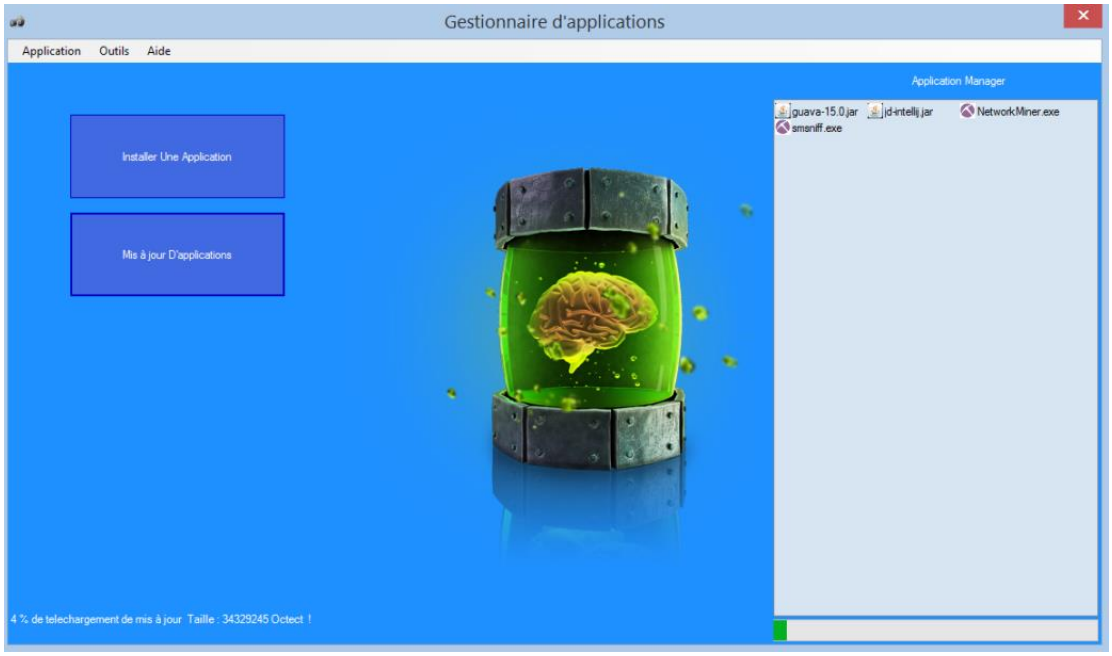


Figure 47 : Gestionnaire d'application en cours d'une mise à jour

Annexe 2 Serveur de la mise à jour

L'interface principale du serveur de mise à jour des applications de la table est la suivante :

Figure 48 : Serveur de Mise à Jour, Interface Principale

Le menu outils contient deux sous-menus: Options et Quitter, comme montre la figure suivante :

Figure 49 : Serveur de Mise à Jour, Menu 2

La figure suivante montre le menu « Help » :

Figure 50 : Serveur de Mise à Jour, Menu 2

Lorsque nous sélectionne une application pour la mise à jour, une interface de la sélection de dossier d'application va apparaitre comme montre la figure suivante:

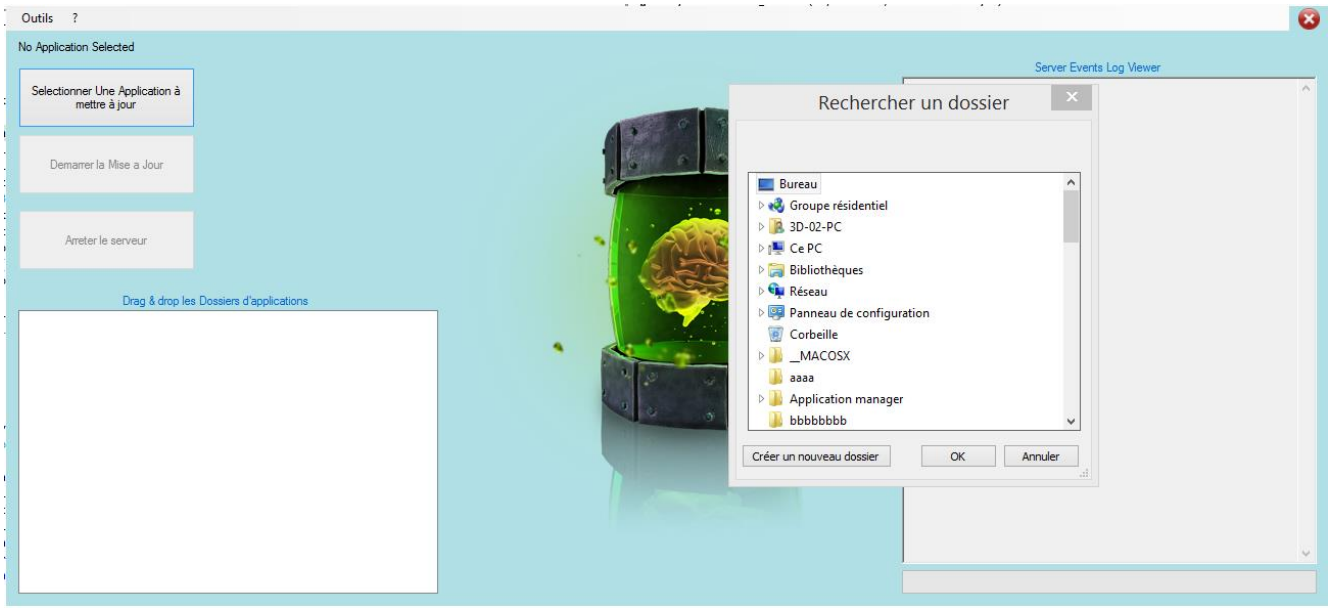


Figure 51 : Serveur de Mise à Jour, Interface d'ajout d'une application

La modification des options du serveur se passe seulement à travers de cette interface qui permet de modifier le numéro du port de la connexion au serveur, comme montre la figure suivante:

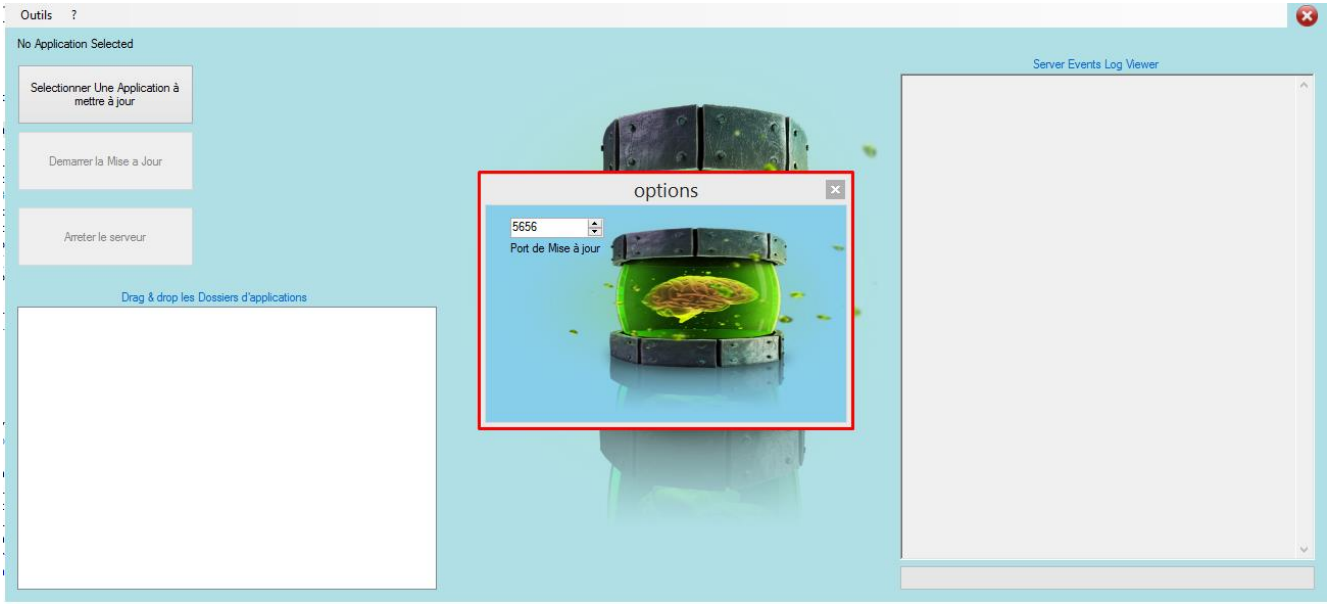


Figure 52 : Serveur de Mise à Jour, Interface Option

La figure suivante montre le processus de la mise à jour en cours (barre de progression et affichage de ficher LOG.txt à l'écran en temps réel) :

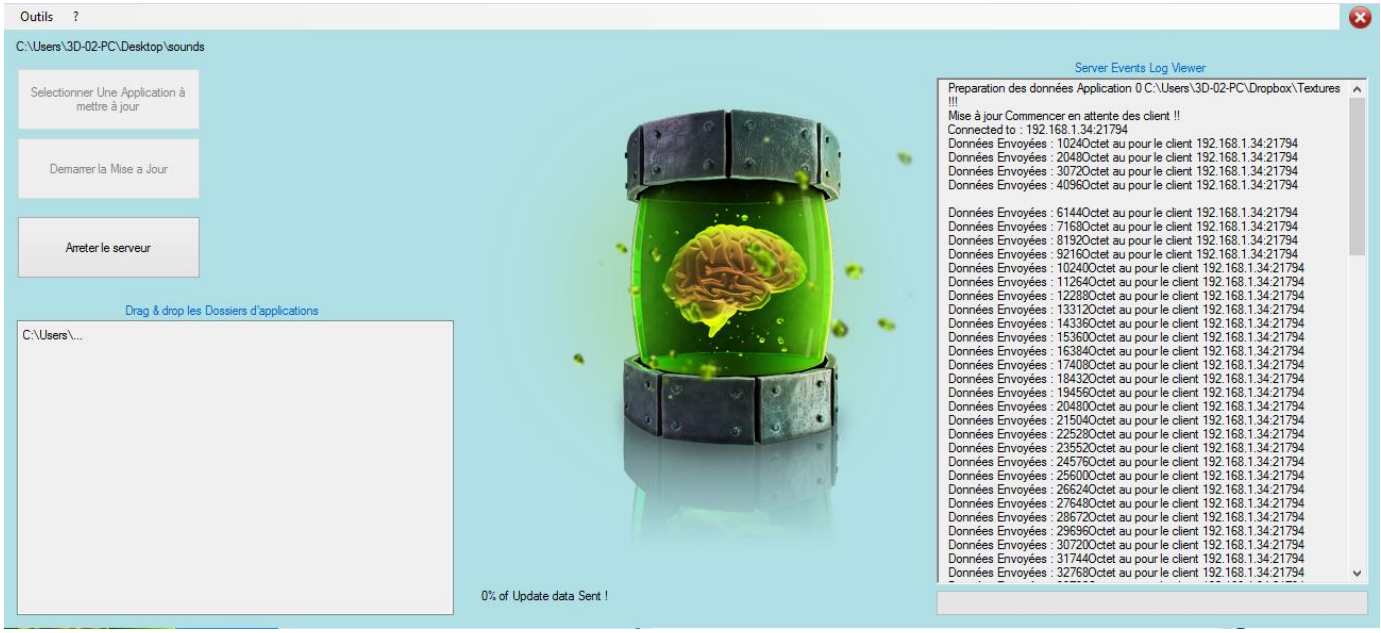


Figure 53 : Serveur de mise à jour en cour d'une mise à jour

Annexe 3 Tangible Framework

1) Aperçu du code Source :

Dans cette partie je vais décrire quelques classes sans afficher le code source de toutes les méthodes.

Pour plus de détails visiter le centre de support pour la documentation:

i. La Classe principale « TableServer.cs » :

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Net;
using System.Net.NetworkInformation;
using System.Net.Sockets;
using System.Net.Configuration;
using System.Text;
using System.Threading;
using System.Runtime.InteropServices;
namespace ColocartsTangibleFramework{
///<summary>
///This is the main Class in this framework, wich contain all the mecanism of network listening, setting of tangible
table (read more about            Framework design)
///<author>Bilel Mnasser</author>
///</summary>
Public class TableServer
    {
///<summary>
/// this static Objeject represent a table and inside it all tags and all slabs and their tags (needed for
add/insert/delete tags)
///</summary>
Public static Table table;
///<summary>
///a static Dictionary contain the Slab IP and her position inside the table
///</summary>
Public staticDictionary<String, Point> SlabPositions ;
///<summary>
///Network Port number for communication with the Tangible Table (65000 default value)
///</summary>
Public static int PortNumber = 65000;
///<summary>
///Network Socket for network communication with the table
///</summary>
Private Socket CommandingSocket;
///<summary>
///This thread will be running for listening on network to capture the table tangible frames
///</summary>
Private Thread ServerThread;
///<summary>
/// Listening Statut ( ON / OFF default)
///</summary>
Private static bool ServerIsRunning = false;
///<summary>
/// Byte Buffer for table command or Reply (1024 Byte default size)
///</summary>
Public static Byte[] MainBuffer = newByte[1024];
///<summary> UDP Listener for reading all table frames Broadcast
///</summary>
UdpClient UDPListner;
///<summary>
/// This Object represent a IP Address
///</summary>
IPEndPoint IpAddress;
///<summary>
/// static Frames Time Out integer (default value 100 milli-seconds)
///</summary>
Public static Int32 TimeOut=100;
///<summary>
/// XML File Path of Slabs IP List
///</summary>
Public static string SlabsIPFile = @"C:\TableFiles\table.xml";
///<summary>
```

```
/// List of All Slabs IP address String
///</summary>
List<String> ConnectedSlabList;
///<summary>
/// Minimum Address Ip for the automatic Network Search (default value 100)
///</summary>
Static public int IPMinimum=100;
///<summary>
/// Maximum Address Ip for the automatic Network Search (default value 255)
///</summary>
Static public int IPMaximum=255;
///<summary>
/// Table RFID Sensor Width (default value 16)
///</summary>
Static public int Table_Sensors_WIdth=16;
///<summary>
/// Table RFID Sensor Height (default value 16)
///</summary>
Static public int Table_Sensors_height = 16;
///<summary>
/// Slab RFID Sensor Width (default value 4)
///</summary>
Static public int Slab_Sensors_WIdth = 4;
///<summary>
/// Slab RFID Sensor Height (default value 4)
///</summary>
Static public int Slab_Sensors_Height = 4;
///<summary>
/// Minimum RSSSI Value for RSSI Detection Mode(default value =0)
///</summary>
Static public int minimum_Rssi_Value=0;
///<summary>
/// RSSI Detection Mode Statut ( ON / OFF default)
///</summary>
Static public bool RSSIModeIsActivated = false;
///<summary>
/// List Of All Tags ID that will be concerned
///</summary>
Static public List<string> TagsID=newList<string>();
///<summary>
/// Event Handler for Tags leaving the table
///</summary>
Public event ChangedEventHandler TagsLeaving;
///<summary>
/// Event Handler for Tags moving on the table
///</summary>
Public event ChangedEventHandler TagsUpdates;
///<summary>
///Virtual Function for  Tags leaving (User must write his own function)
///<param name="sender">the tag object that rise the event</param>
///<param name="e">the event rised by the sender</param>
///<return> this function return type is void </return>
///</summary>
Protected virtual void OnTagsLeaving(Tag sender, EventArgs e){};
///<summary>
///Virtual Function for  Tags Moving (User must write his own function)
///<param name="sender">the tag object that rise the event</param>
///<param name="e">the event rised by the sender</param>
///<return> this function return type is void </return>
///</summary>
Protected virtual void OnTagsMoving(Tag sender,EventArgs e){};
///<summary>
///Default Constructor, Wich will initialize all class members
///</summary>
public TableServer(){};
///<summary>
/// this function will inisialize all class members (buffers, objects, Network listeners ... etc)
///<return> this function return type is void </return>
///</summary>
Private void initilize_Class_Members(){};
///<summary>
///private Function for reading Slabs Ip from XML file.
///<return> this function return type is void </return>
///</summary>
Private void Configure_List_Of_Slabs_From_XML(){};
///<summary>
///Private Function for automatic setting by sending comands
///for each Slab on the tangible table (IP server, detection Mode... etc)
```

```
///<return> this function return type is void </return>
///
///</summary>
Private void Configure_Parametres_des_Slabs(){} ;
///<summary>
///Private Function for initializing Slabs positions on the tangible table
///<return> this function return type is void </return>
///</summary>
Private void Configure_Positions_OfSlabs(){};
///<summary>
///Public Function for Start Listening on Network for Broadcast tangible frames
///<return> this function return type is void </return>
///<example>TableServer tab=new tableServer(); tab.start(); </example>
///</summary>
Public void Start(){};
///<summary>
///Public Function for Stop Listening on Network for Broadcast tangible frames
///<return> this function return type is void </return>
///<example> TableServer tab=new tableServer(); tab.start(); tab.Stop() </example>
///</summary>
Public void Stop(){};
///<summary>
///private Function for performing the listening on Network Broadcast tangible frames
///<return> this function return type is void </return>
///</summary>
Private void Run(){};
///<summary>
///Static Public Function for Stop Listening on Network for Broadcast tangible frames
///<return> this function return type is List of String </return>
///<example>Foreach(string s in TableServer.AutomaticNetworkSearch())
///{Console.WriteLine(s); };
///</example>
///</summary>
Public staticList<String> AutomaticNetworkSearch(){};
///<summary>
/// Public Function to send a simple Tangible Command to a Single Slab
///<param name="IPSlab">the string value of the Slab IP address</param>
///<param name="Cmd">the string value of the Tangible Command</param>
///<return> this function return type is Response Object Type</return>
///<example>TableServer tab= new TableServer();
/// tab.SendCommand("192.186.1.101","mode:1");
///</example>
///</summary>
Public Response SendCommand(string IPSlab, String Cmd){};
///<summary>
/// Private Function to get a position of tag from his ID Value
///<param name="IDRFID">the string value Tag RFID Identificator</param>
///<return> this function return type is tag Object Type</return>
///</summary>
Private staticTag Get_Tag_Position(String IDRFID){};
///<summary>
/// public Function to calculate the medium point of list of points
///<param name="ListPoint">the List of points objects</param>
///<return> this function return type is Point Object Type</return>
///</summary>
Public staticPoint calcul_Midium_point(List<Point> ListPoint){};
        ///<summary>
/// public Function to Liberate All Resources and clean everything
///<return> this function return type is void</return>
///</summary>
Public void Dispose(){} ; }
  }
```

ii. *Classe « Serialization.cs » :*

Cette classe (avec des méthodes qui rend une valeur booléenne comme valeur de retour) sert à lire ou écrire dans un fichier XML la liste des adresses IP des dalles de la table tangible sous la forme suivante (la liste de dalles commence par le haut-gauche):

```
<?xmlversion="1.0"encoding="utf-8"?>
<ArrayOfStringxmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<string>192.168.1.111</string>
<string>192.168.1.112</string>
</ArrayOfString>
```

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Xml.Serialization;
namespace ColocartsTangibleFramework
{///<summary>
/// This class used for Serialization/Deserialization of any list object into a file on hard disk.
///</summary>
///<author>Bilel Mnasser</author>
PublicstaticclassSerialization
   {     ///<summary>
///static function to Serialize a list into a file on hard disk.
///<param name="List">list to Deserialize.</param>
///<param name="filename">File path.</param>
///<return> this function return type is bool. </return>///</summary>
Public static bool SerializeObject(refList<String> List, String filename)
        {try{   //objet de serialisation
XmlSerializer xmlSerializer = newXmlSerializer(List.GetType());
//ouverture de flux d'ecriture dans un fichier
TextWriter textWriter = newStreamWriter(filename);
//serialisation d'object { paramètres : 1.flux de fichier 2.object List}
xmlSerializer.Serialize(textWriter, List);
//fermeture de flux
                textWriter.Close();
//serialization not failed
Return true;
            }
catch (Exception e)
            {   //write the exception in terminal
Console.WriteLine(e.StackTrace);
//serialization failed return false
Return false;
            }
          }
///<summary>
///static function to Deserialize a list from a file on hard disk.
///<param name="List">list to Deserialize.</param>
///<param name="filename">File path.</param>
///<return> this function return type is bool. </return>///</summary>
Public static bool DeserializeObject(refList<String> List, String filename)
        {try{//objet de serialisation
XmlSerializer xmlSerializer = newXmlSerializer(List.GetType());
//ouverture de flux de lecture d'un fichier
TextReader textReader = newStreamReader(filename);
//desrialisation de l'objet{ a partir d'un flux de fichiser et une conversion de type est nécessaire}
List = (List<String>)xmlSerializer.Deserialize(textReader);
//fermeture de flux
             textReader.Close();
//serialization not failed
Return true;
            }
catch (Exception e)
            {   //write the exception in terminal
Console.WriteLine(e.StackTrace);
//serialization failed return false
Return false;
            }

        }
    }
}
```

2) Exemple d'utilisation du« Tangible Framework » en c# et Visual studio :

Voici un simple exemple téléchargeable ici, qui va afficher sur l'invite de commande des informations (ID et positions) des tags, En utilisant mon framework:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using TangibleFramework;
namespace test
{
classProgram
    {
Static void Main(string[] args)
        {   //inistialize server
TableServer tableserver = newTableServer();
//add the id tag that we want to the server manage his events
TableServer.List_Tags_ID.Add("7197860FED2002E0");
//connect our function to the events handlers
tableserver.Tags_Updates+=tableserver_Tags_Updates;
tableserver.Tags_Leaving+=tableserver_Tags_Leaving;
//we start the listening at the network
tableserver.Start();
}

Private static void tableserver_Tags_Leaving(Tag sender, EventArgs e)
{//write on console the tag ID
Console.WriteLine("the Tag with ID=" + sender.ID + "has left the table now");
 }

Private static void tableserver_Tags_Updates(Tag sender, EventArgs e)
{//write on console tag ID position
Console.WriteLine("the Tag with ID="+sender.ID + " at this position now :" + sender.Position.X+"
"+sender.Position.Y);
}


    }
}
```

Le résultat de cet exemple lorsqu'on pose un tag sur la table pour 2 secondes et qu'on l'enlève est le suivant:

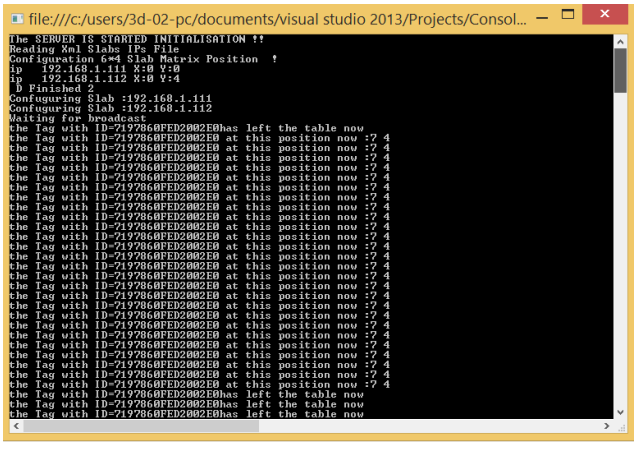


Figure 54:résultat d'exemple d'utilisation« Tangible Framework »

Il faut tout d'abord ajouter le DLL à la liste de références de votre projet tangible, comme montre la figure suivante :

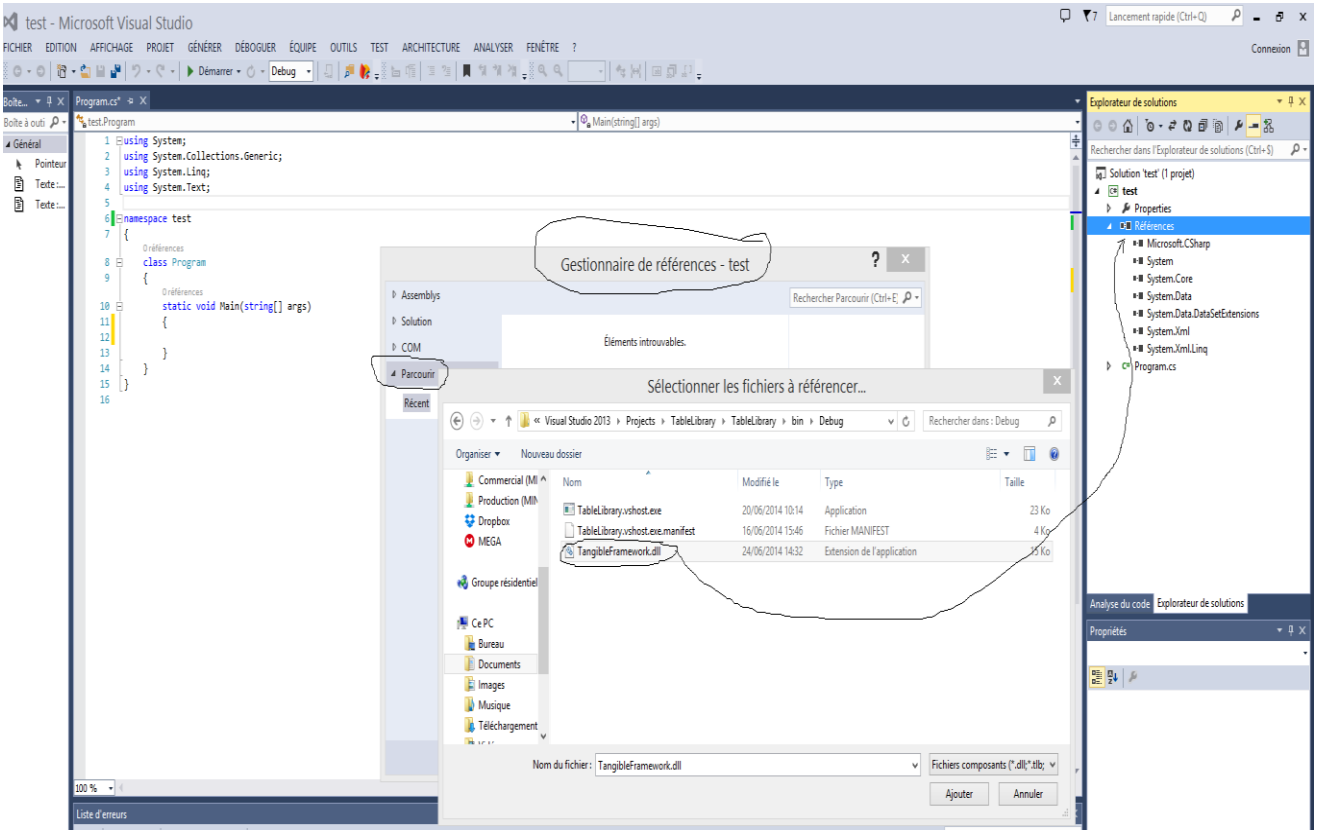


Figure 55 : Tutoriel d'importer la framework

3) Applications de démonstration pour « Tangible Framework »:

Après le développement et la documentation de notre API qui nous permet de se connecter à la table tangible, j'ai développé des petites applications de démonstration pour montrer et tester notre Api, ces dernières jouent le rôle de tutoriels de démonstration d'utilisation de notre Api, pour aider les nouveaux développeurs a bien comprendre l'utilisation de notre API. Ces applications sont développées dans différentes technologies tel que JAVA et C# (Eclipse, Unity 3D …). Dans ces applications, les objets graphiques suivent les déplacements de leurs tags correspondants sur la table.

a) Colored Tags Demo:

Une petite application qui montre l'utilisation de notre api, développé en java sous Eclipse Kepler 2014, qui montre tous les puces RFID de la table sous forme d'un tableau visuel.

A chaque nouveau tag détecté par une puce, la case correspondante dans le tableau réagit en changeant la couleur, une application qui montre que L'API est accessible par différentes technologies (Java par exemple).

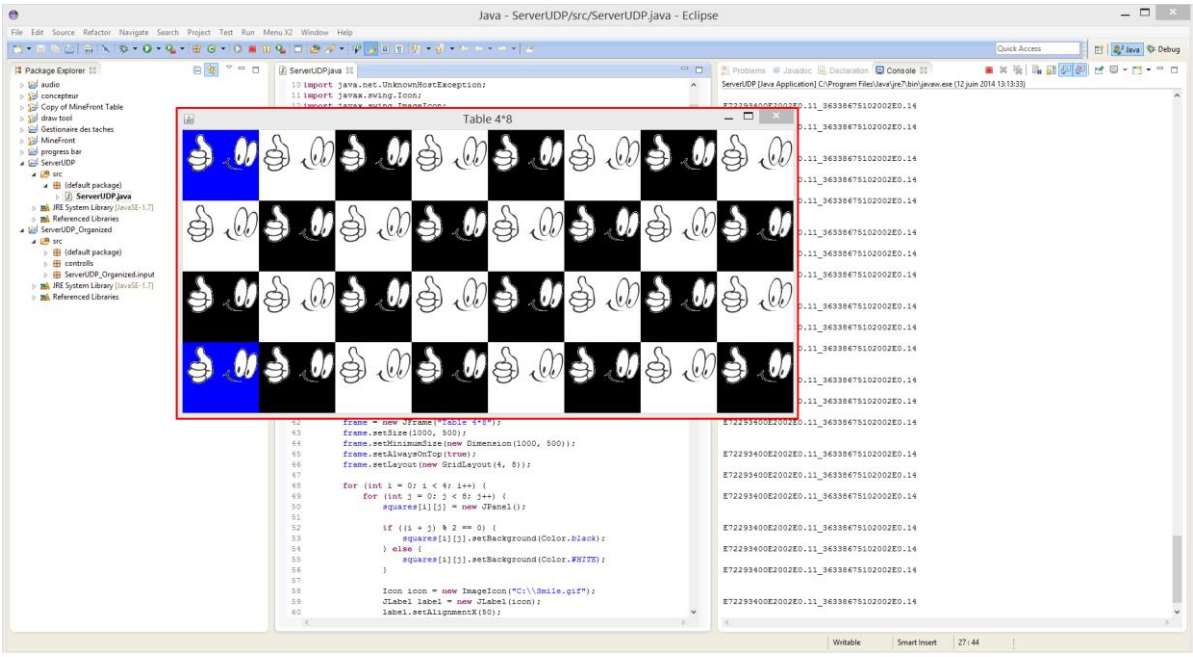


Figure 56 : Colored Tags Demo

b) Tag Light 2D Demo:

Une application développée sous Unity3D version 4.5 professionnel, en programmation c#, qui montre des interactions entre trois Tags détectés par la table, (un tag représente la lumière, un autre représente un cube, un autre représente un cylindre). Dans cette application, des ombres vont être dessinées lors de rapprochement de tag lumière vers le tag cube ou le tag cylindre.

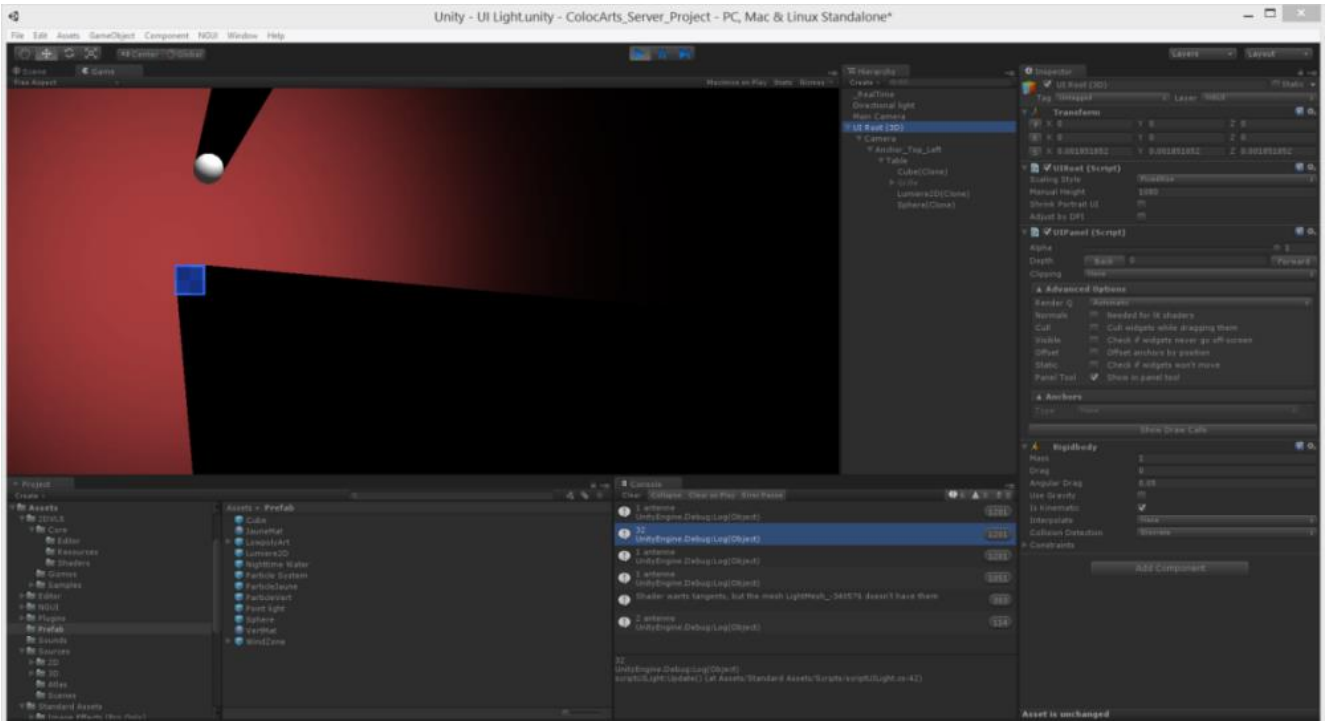


Figure 57 : Tag Light 2D Demo

c) *Particles system Demo:*

Une application développée sous Unity 3D version 4.5 professionnel, en programmation c#, qui montre des interactions en 3D entre trois Tags détectés par la table, (un tag représente objet respirateur, un autre représente un émetteur de particules jaunes et un dernier représente un émetteur de particules vertes). Lors de rapprochement du tag respirateur aux autres tags, des interactions 3D vont se dérouler.

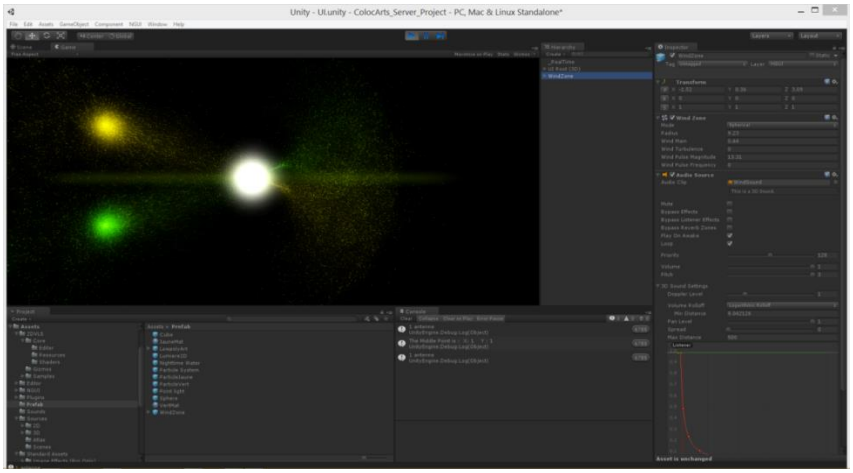


Figure 58 : System Particles Demo

d) *3D animal parc Demo:*

C'est une application développée sous Unity3D version 4.5 professionnel, en programmation c#, qui montre des interactions en 3D entre trois Tags détectés par la table, (un tag représente ours, un autre représente un cerf, un autre représente aigle royale). Le cerf va être mangé par l'ours et l'aigle Royale si on rapproche leurs tags.

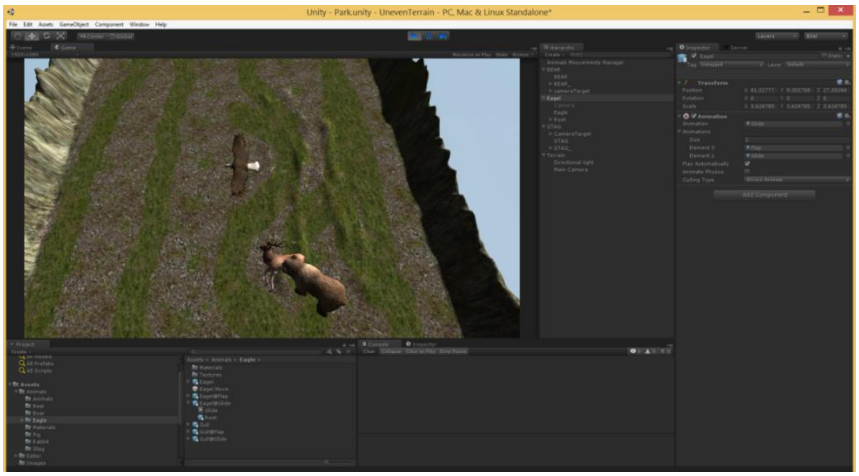


Figure 59 : 3D animal parc Demo

Annexe 4 Détails de la table tangible

Les figures suivantes montrent la partie tangible de notre table :

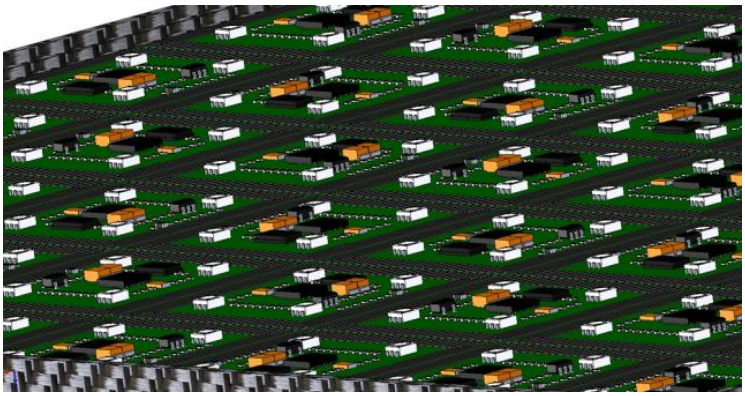


Figure 60 : Vue dalle tangible

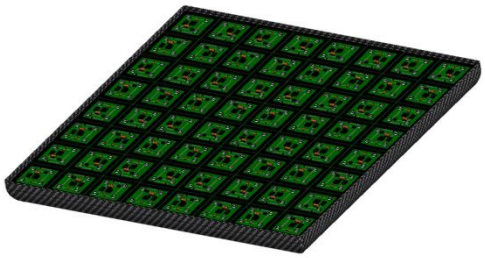


Figure 61 : Table Tangible vue dessus

Figure 62 : Table tangible Vue de coté 1

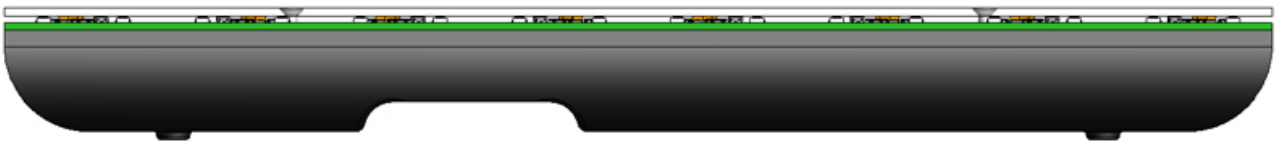


Figure 63 : Table tangible Vue de coté 2

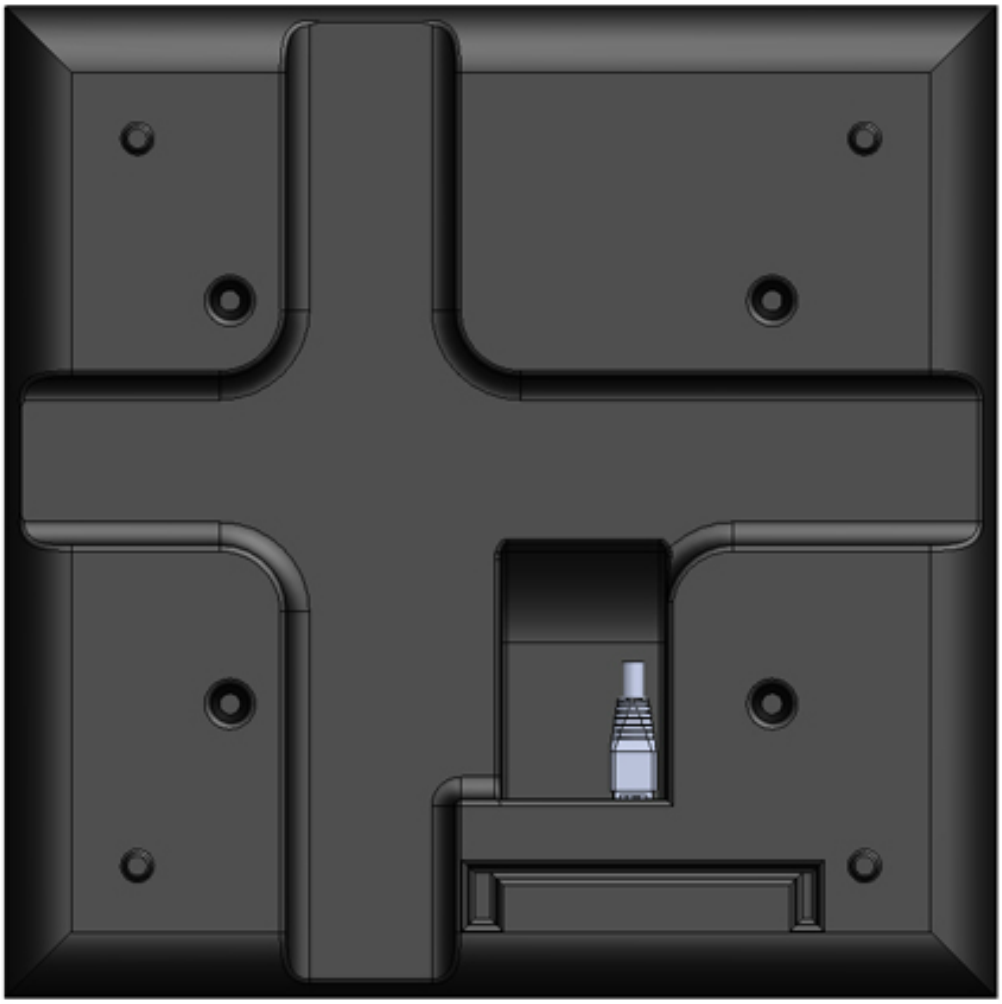


Figure 64 : Table tangible Vue de dessous

Annexe 5 Biodiversité aux bouts des doigts version 2

Dans cette annexe je vais donner une description sur la programmation du jeu, les commentaires des codes sont rédigés en anglais pour des raisons de documentation. Notre jeu se compose d'un nombre des composants graphiques :

- Des animaux.
- Un Plan.
- Des zones géographiques.
- Des objets d'informations pour le score et chaque animal ou zone.

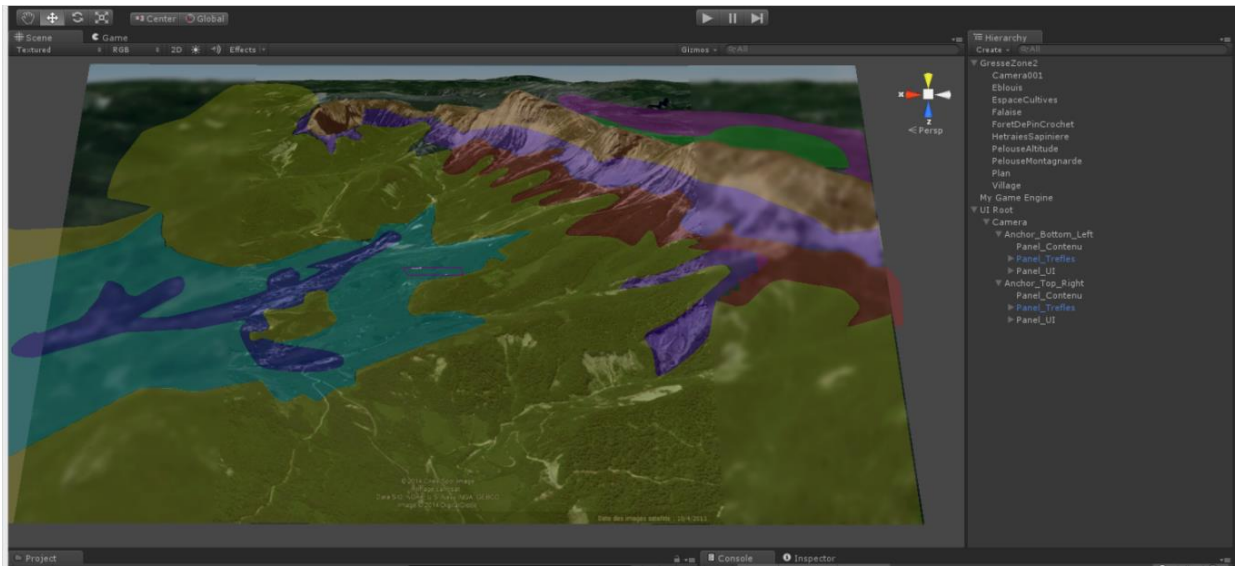


Figure 65 : Unity Game Objects

1) Plan:

Représente la carte principale du Vercors, il comporte des composants graphiques enfants des différentes zones de la carte du Vercors :

Figure 66 : Game Object plan

Chaque zone Contient un composant **Mesh-Collider** (qui sert à la détection de la collision avec les animaux, la loupe …) et un composant **Mesh-Renderer** qui sert à dessiner et afficher l'objet sur l'écran.

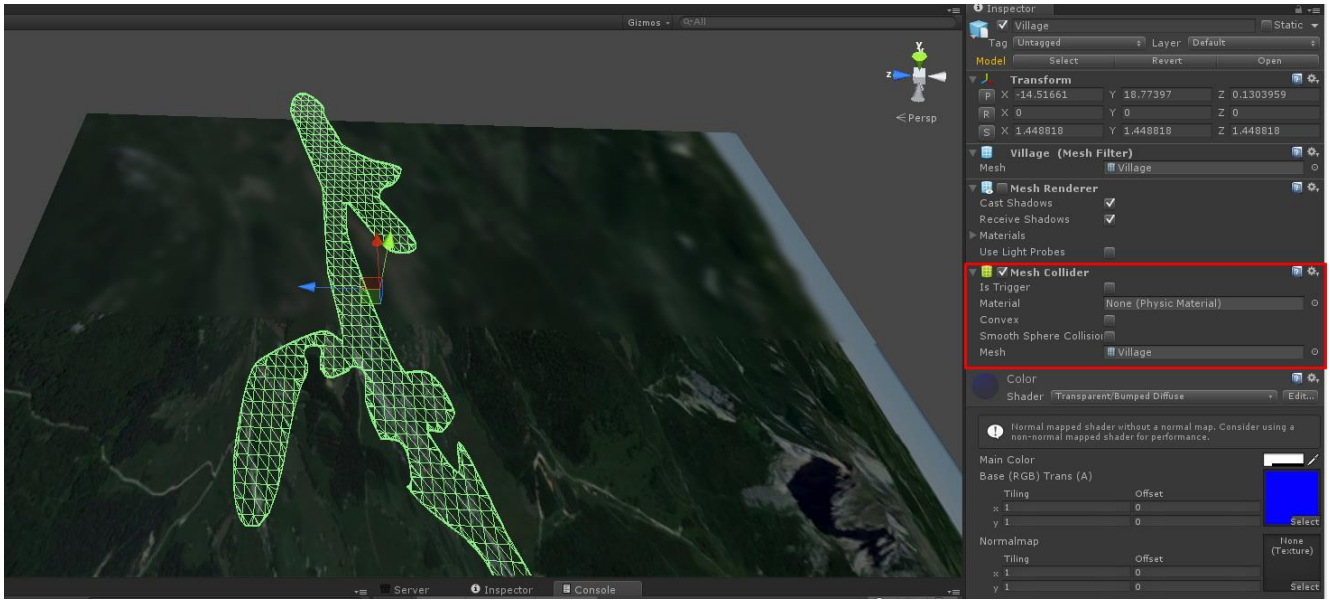


Figure 67 : Zone Village Mesh Collider, Zone Village Mesh Renderer

2) Prefabs des animaux:

Nos Prefabs animaux comportent les différents composants suivant :

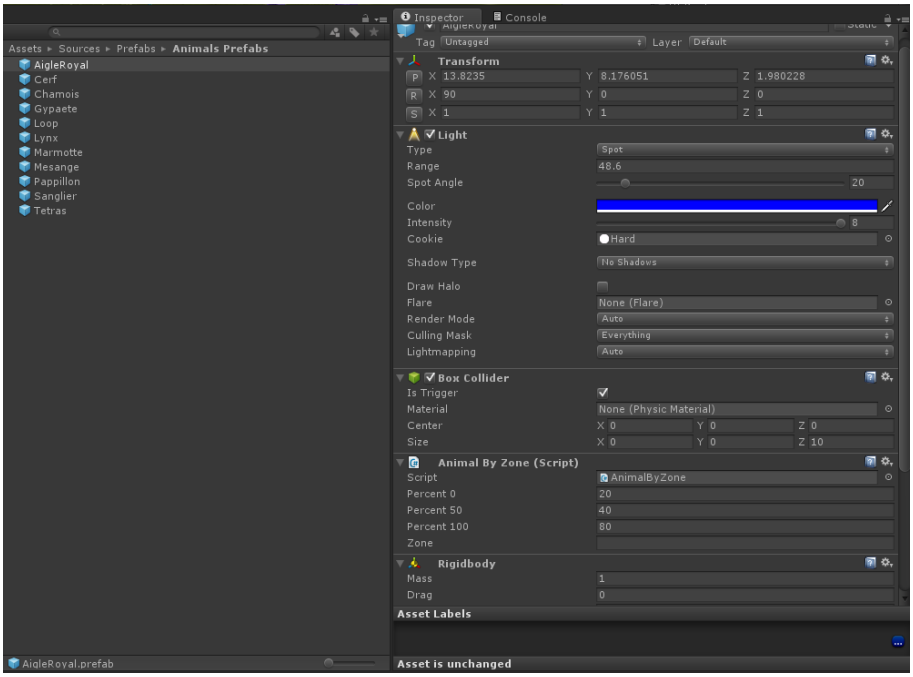


Figure 68 : Animal Game Objects

1. **Composant Light**: c'est un composant principal pour nos Animaux qui gère la zone de domination de l'animal sur la carte.

2. **Composant Box Collider** : Un collisionneur primitif en forme de boîte, un composant standard d'Unity 3D pour les collisions.
3. **Composant Renderer** : c'est un composant principal pour l'affichage de l'objet sur l'écran.
4. **Composant Rigidbody** : Ajout d'un composant Rigidbody à un objet mettra son mouvement sous le contrôle du moteur physique d'Unity 3D (détection des collisions par exemple).
5. **Composant Animal By Zone (Script)** : un script écrit en c# pour la mise à jour de la propriété « ***Spot Angle*** » de Composant « ***Light*** » en se basant sur la collision de « ***Box Collider*** » avec les différentes zones dans l'objet « Plan » :

```
using UnityEngine;
using System.Collections;

public class AnimalByZone : MonoBehaviour
{
//Defenir les valeur de pourcentage de chaque animal dans les diffrentes zones
    # region PercentValue
Public float Percent0 = 20;
Public float Percent50 = 40;
Public float Percent100 = 80;
    #endregion
//recuperer la zone actuelle de chaque animal
#region Zone
Public string Zone="";
#endregion
//this function will be executed on every collision, and will set for each animal the radius of his
domination depending on animal type
void OnTriggerStay(Collider other){} ;

}
```

3) La Loupe :

La loupe comporte les composants suivants :

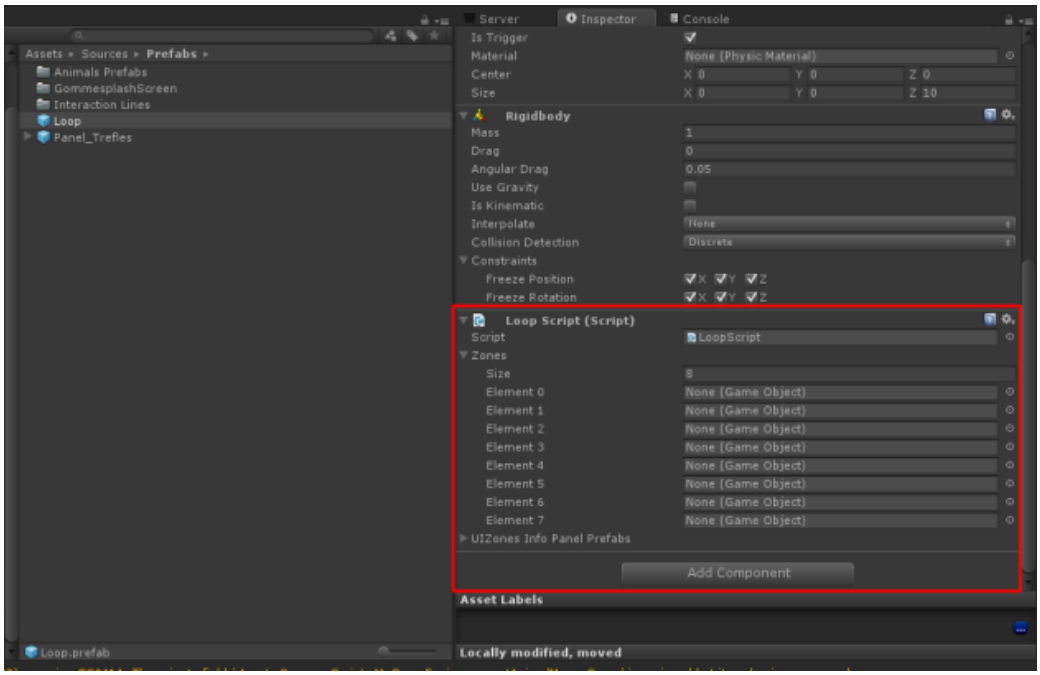


Figure 69 : Game Object Loupe

1. **Composant Box Collider** : Un collisionneur primitif en forme de boîte, un composant standard d'Unity 3D pour les collisions.
2. **Composant Renderer** : c'est un composant principal pour l'affichage de l'objet sur l'écran.
3. **Composant Loupe (Script)** : un script écrit en c# pour la mise à jour de la propriété « ***enabled*** » de Composant « ***Renderer*** » des différentes zones de l'objet « **Plan** ». en se basant sur la collision avec les Colliders des zones :

```
using UnityEngine;
using System.Collections;
[System.Serializable]
Publicclass UIZonesInfoPanelprefabs
public UIPanel village;
public UIPanel EspaceCultives;
public UIPanel Sapinaire;
public UIPanel PelouseMontagnarde;
public UIPanel Eboulis;
public UIPanel Falaise;
public UIPanel ForetDePinCrochet;
public UIPanel PelouseAltitude;

}
Publicclass LoupeScript : MonoBehaviour {
    [SerializeField]
//defenir les gameobjects des zones de notre carte
        publicGameObject[] Zones;
    [SerializeField]
//defenir les gameobjects des Informations des zones de notre carte
Public UIZonesInfoPanelprefabs UIZonesInfoPanelPrefabs;
void OnTriggerStay(Collider other)
{
//cacher toutes les zones lors d'une nouvelle collision
foreach (GameObject t in Zones)
{
                    t.renderer.enabled = false;

             }
//afficher la zone en collision avec la loupe
other.gameObject.renderer.enabled = true;

}

}
```

4) UI root:

C'est un objet qui contient les différents objets d'affichage d'informations concernant les zones, les animaux et le score. Cet objet représente deux panneaux (gauche et droite) avec des positions fixes qui servent à afficher les informations.

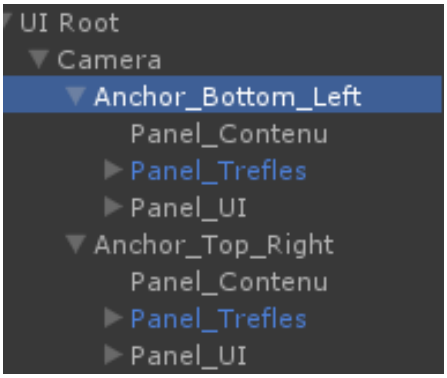


Figure 70 : UI root Game Object

5) Définition d'un Identificateur RFID pour chaque Game Object :

Une classe statique pour définir un identificateur tangible (RFID) pour chaque objet du jeu.

```
using UnityEngine;
using System.Collections;

        public static classTag {
        public static string Gypaete = "7297860FED2002E0";
        public static string Lynx = "7497860FED2002E0";
        public static string AigleRoyal = "6F98860FED2002E0";
        public static string Sanglier = "7298860FED2002E0";
        public static string Marmotte = "6D98860FED2002E0";
        public static string Tetras = "7198860FED2002E0";
        public static string Cerf = "7597860FED2002E0";
        public static string Chamois = "7098860FED2002E0";
        public static string Mesange = "7397860FED2002E0";
        public static string Pappillon = "7398860FED2002E0";
        public static string Loupe = "7197860FED2002E0";
        public static string Gomme = "6E98860FED2002E0";
}
```

6) Vercors Game Engine :

L’objet principal qui contient le script principal du jeu, le script prend en entrée quelques paramètres:

Figure 71 : paramètres du Game Engine inspecteur

Il comporte toutes les fonctions suivantes qui gèrent le comportement tous les objets du jeu :

```
//function for initiliasing all vercors Game Engine script parameters.
Public void InitialiseGameEngine(){}
//fix All the current Animal at their position when a new animals detected in affection zone !
void FixPreviousAnimals(){}
// update the score
Private void Update_Score(){}
//my delegate function to update position from table
Private void On_Tags_Updated_On_Table(Tags sender, System.EventArgs e){}
//my delegate function when tags leave the table
Private void On_Tag_Deleted_On_Table(Tags sender, System.EventArgs e){}
// initialize all connection to tangible table (Initialize Colocarts tangible Framework)
void Start()
{
InitialiseGame();
//cacher le curseur de la souris
Screen.showCursor = false;
//instantite my server
server = newTableServer();
// On monte les deux fonctions délègues sur nos deux évent Handler;
//pour la détection de mouvement de tags de la table ou leurs absence
server.TagsNotOnTable += newChangedEventHandler(On_Tag_Deleted_On_Table);
// pour savoir les mis a jour des positionement
server.TagsUpdates += newChangedEventHandler(On_Tags_Updated_On_Table);
//on demarre le serveur sur la table
server.Start();
        }
// Funtion will be called every frame while the game is running (this function will call all those function every
frame)
void Update()
        {
            Loupe();
            Gypaete();
            AigleRoyal();
            Lynx();
            Pappillon();
            Sanglier();
            Marmote();
            Cerf();
            Mesange();
            Tetras();
            Chamois();
            Animals_Interactions();
            Gomme();
}
//draw 1 interaction between 2 position, color of interaction, number
void DrawInteraction(int i, Vector3 startPosition, Vector3 EndPosition, Color color) {}
//draw all animal instanciated in the game intercations
Private void Animals_Interactions(){}
//Function of Gomme game Object, instanciate, restart the game and destroy him when he leave the table
Gomme(){}
//Function of Loupe game Object, instanciate, move position and destroy him when he leave the table
Loupe(){}
//Function of Gypaete game Object, instanciate, move position and destroy him when he leave the table
void Gypaete() {}
//Function of Lynx game Object, instanciate, move position and destroy him when he leave the table
void Lynx(){}
//Function of AigleRoyal game Object, instanciate, move position and destroy him when he leave the table
void AigleRoyal(){}
//Function of Sanglier game Object, instanciate, move position and destroy him when he leave the table
void Sanglier(){}
//Function of Marmote game Object, instanciate, move position and destroy him when he leave the table
void Marmote(){}
//Function of Tetras game Object, instanciate, move position and destroy him when he leave the table
void Tetras(){}
//Function of Cerf game Object, instanciate, move position and destroy him when he leave the table
void Cerf(){}
//Function of Mesange game Object, instanciate, move position and destroy him when he leave the table
void Mesange(){}
//Function of Pappillon game Object, instanciate, move position and destroy him when he leave the table
void Pappillon(){}
// destroy Gypaete animal
```

```
Public IEnumerator DestroyGypaete(float number){}
// destroy Lynx animal
Public IEnumerator DestroyLynx(float number){}
// destroy Aigle Royal animal
Public IEnumerator DestroyAigleRoyal(float number){}
// destroy Sanglier animal
Public IEnumerator DestroySanglier(float number) {}
// destroy Marmotte animal
Public IEnumerator DestroyMarmotte(float number) {}
// destroy Tetras animal
Public IEnumerator DestroyTetras(float number){}
// destroy Cerf animal
Public IEnumerator DestroyCerf(float number){}
// destroy Chamois animal
Public IEnumerator DestroyChamois(float number){}
// destroy mesange animal
Public IEnumerator DestroyMesange(float number){}
//destroy pappillion animal
Public IEnumerator DestroyPappillon(float number){}
//destroy loupe object
Public IEnumerator DestroyLoupe(float number){}
//Function that will Liberate resources and close connection with table when the game colsing
void OnApplicationQuit(){}
```

1. Format général des trames

Tous les octets composants les trames sont au format ASCII.

Le format des trames suit le modèle :

[STX]	commande	:	corps [ETX]		[horadatage]

S'il s'agit d'une réponse, [ACK] (commande traitée avec succès) ou [SYN] (échec du traitement de la commande) sont ajoutés au début de la trame :

[ACK] ou [SYN]	[STX]	commande	:	corps [ETX]		[horodatage]

L'horodatage est au format : HHMMSSmmm (heures, minutes, secondes, milli-secondes).

Les commandes et corps des trames peuvent être amenés à changer.

Version ≤ 6

Certaines commandes commencent par un 'X'. Dans ce cas, le 'X' doit être remplacé par un numéro sur 1 caractère qui correspond au numéro de la carte RFID. Une dalle RFID est composée de 4 cartes RFID. Voici l'emplacement des cartes RFID :

1	2
3	4

Placement des cartes RFID sur une dalle

Chaque carte RFID contient 4x4=16 antennes. Ainsi la carte numéro 1 est associée aux antennes telles que : 0 < x <5 et 0 < y <5. La carte numéro 2 est associée aux antennes telles que : 4 < x <9 et 0 < y <5, etc.

Si 'X' est remplacé par '0', alors la commande est transmise à toutes les cartes RFID.

Version ≥ 7

A partir de la version 7, il n'y a plus qu'un seul microcontrôleur RFID. Pour s'adresser à ce microcontrôleur, il faut précéder les commandes d'un numéro, comme s'il s'agissait des cartes RFID.

2. Détail des trames

Les commandes en italique ne sont pas encore supportées.
Lors de la mise sous tension, la dalle est configurée avec les paramètres par défaut.

2.1. Commandes de paramètres de fonctionnement

Ces commandes servent à transmettre des paramètres à la dalle.

Commande	**Format du corps**	
mode	Question	0 : mode Question-Réponse (défaut) 1 : mode Autonome/Etat complet 2 : mode Autonome/Transition
	Réponse	idem
	Commentaires	Cette commande définit le mode de fonctionnement de la dalle. En mode autonome, la dalle lance elle-même les séquences d'inventaire, et envoie les trames à l'adresse IP spécifiée par la commande “setserverip”. Même si le mode autonome est activé, la dalle répond quand même aux questions qui lui sont posées. Cependant les commandes suivantes renverront une erreur : Sid, presence, read, write, optimise, synchro, tackeys, nbantennes, nbcartes
RSSI	Question	Chiffre entre 0 et 7 inclus (défaut = 0)
	Réponse	idem
	Commentaires	Ce paramètre n'est utile qu'en mode Autonome/Transition. Il réprésente la variation minimale du RSSI qui est considérée comme transition. Une valeur de 7 signifie qu'il n'y a transition seulement lors d'apparition ou de disparition de tag (du point de vue d'une antenne).
delai	Question	Valeur (entre 000 et 999) sur 3 caractères (défaut = 000)
	Réponse	idem
	Commentaires	Délai minimal entre la réception d'une commande et sa réponse (en millisecondes).
heure	Question	HHMMSS (défaut = 0 à la mise sous tension)
	Réponse	idem
	Commentaires	Cette commande sert à mettre chaque dalle à l'heure. Les millisecondes sont mises à zéro.
RFID	Question	0 : Protocole ISO15693 (défaut) 1 : Protocole ISO14443A 2 : Protocole ISO14443B
	Réponse	idem
	Commentaires	Spécifie le protocole RFID.

	Réponse	idem
	Commentaires	Ce paramètre n'est utile qu'en mode Autonome et spécifie le format de trame utilisé en mode Autonome.
horodatage	Question	0 : Horodatage désactivé 1 : Horodatage activé (défaut)
	Réponse	idem
	Commentaires	Active ou désactive l'horodatage des trames.
optimise	Question	Version ≤ 6 0 : Optimisations désactivées : 4 sur 64 (défaut) 1 : Optimisations X2 activées : 8 sur 64 2 : Optimisations X4 activées : 16 sur 64 Version ≥ 7 0 : Optimisations désactivées : 1 sur 64 1 : Optimisations activées : 2 sur 64 2 : Optimisations activées : 4 sur 64 3 : Optimisations activées : 8 sur 64 4 : Optimisations activées : 16 sur 64
	Réponse	idem
	Commentaires	Ces optimisations augmente la vitesse des inventaires (commandes "presence" et "sid", et mode autonome) en activant plusieurs antennes à la fois. Version ≤ 6 En mode autonome synchronisé, les fréquences d'inventaire sont : - sans optimisations : 7Hz - optimisations X2 : 12Hz - optimisations X4 : 20Hz **(Soucis de fonctionnement)** Version ≥ 7 Les optimisations marchent correctement jusqu'au niveau 1, et un peu moins bien pour les niveaux suivants. La valeur est mémorisée en EEPROM avec la commande "progeeprom".
getposition	Question	(vide)
	Réponse	X . Y X et Y : 1 caractère
	Commentaires	Donne la position de la dalle. Si aucune position n'a été affectée, la réponse est : "0.0".
setposition	Question	X . Y X et Y : 1 caractère
	Réponse	idem
	Commentaires	Permet d'associer une position X,Y à la dalle. Ces valeurs sont mémorisées en EEPROM automatiquement.
		1 : Synchronisation activée (défaut)

	Commentaires	Active ou désactive la synchronisation. Il est recommandé de ne pas la désactiver si plusieurs dalles sont côte-à-côte.
progeeprom	Question	(vide)
	Réponse	(vide)
	Commentaires	Programme la mémoire EEPROM de la dalle avec les variables affectées par les commandes :setposition, optimise, setip, setserverip, setgatewayip, portUDP, portstartUDP et leddiviseur.

2.2. Commandes de configuration réseau

Ces commandes permettent de configurer le réseau.

Commande	Format du corps	
transport	Question	0 : UDP (défaut) 1 : TCP
	Réponse	idem
	Commentaires	Permet de choisir le protocole utilisé pour le mode autonome.
portUDP	Question	Numéro du port sur 5 caractères (défaut = 65000)
	Réponse	idem
	Commentaires	Spécifie le numéro du port UDP du PC pour le mode autonome.
portstartUDP	Question	Numéro du port sur 5 caractères (défaut = 65000)
	Réponse	idem
	Commentaires	Spécifie le numéro du port UDP du PC pour l'envoi de la commande “start”.
portTCP	Question	Numéro du port sur 5 caractères
	Réponse	idem
	Commentaires	Spécifie le numéro du port TCP du PC.
getip	Question	(vide)
	Réponse	XXX.XXX.XXX.XXX (valeurs en décimal)
	Commentaires	Donne l'adresse IP de la dalle.
setip	Question	XXX.XXX.XXX.XXX (valeurs en décimal)
	Réponse	idem
	Commentaires	Permet de changer l'adresse IP de la dalle.
getserverip	Question	(vide)
	Réponse	XXX.XXX.XXX.XXX (valeurs en décimal)
	Commentaires	Donne l'adresse IP du PC vers lequel les données sont envoyées en mode autonome.
setserverip	Question	XXX.XXX.XXX.XXX (valeurs en décimal)
	Réponse	XXX.XXX.XXX.XXX (valeurs en décimal)
	Commentaires	Permet de changer l'adresse IP du PC vers lequel les données sont envoyées en mode autonome.
getgatewayip	Question	(vide)
	Réponse	XXX.XXX.XXX.XXX (valeurs en décimal)
	Commentaires	Donne l'adresse IP de la passerelle. .
setgatewayip	Question	XXX.XXX.XXX.XXX (valeurs en décimal)

Pour que les nouvelles valeurs de portUDP, portstartUDP, portTCP, setip, setserverip et setgatewayip soient mémorisées lors du redémarrage de la dalle, il faut ensuite utiliser la commande “progeeprom”.

Seul le port 65000 (UDP et TCP) des dalles est ouvert.

2.3. Commandes de tags

Ces commandes servent à connaître la position des tags et de lire et écrire dans leur mémoire. Liste des commandes et du corps correspondant :

<table>
<tr><th>Commande</th><th colspan="2">Format du corps</th></tr>
<tr><td rowspan="3">presence</td><td>Question</td><td>(vide)</td></tr>
<tr><td>Réponse</td><td>\n | 1ère ligne | \n | 2ème ligne | etc</td></tr>
<tr><td>Commentaires</td><td>Pour des raisons de lisibilité lors d'une capture de trame, les marqueurs sont des retour à la ligne ('\n'), et le premier caractère est aussi un marqueur. Chaque ligne comporte 8 caractères (0 ou 1), et il y a 8 lignes en tout. Ce format est susceptible d'être modifié.</td></tr>
<tr><td rowspan="3">sid</td><td>Question</td><td>(vide)</td></tr>
<tr><td>Réponse</td><td>SID1 + coordonnées | _ | SID2 + coordonnées | (etc)
Détail d'un SID + coordonnées :
SID | . | X1 | Y2 | R1 | . | X2 | Y2 | R2 | (etc)
SID : SID du tag LSB first (plusieurs octets)
X et Y : coordonnées de l'antenne (entre 1 et 8)
R : RSSI (entre 0 et 7)</td></tr>
<tr><td>Commentaires</td><td>Donne les SID des tags et leur position.</td></tr>
<tr><td rowspan="3">read</td><td>Question</td><td>X | Y | N
X : coordonnée en X sur 1 caractère
Y : coordonnée en Y sur 1 caractère
N : numéro du bloc sur 2 caractères</td></tr>
<tr><td>Réponse</td><td>X | Y | N | D
D : données du bloc mémoire (8 caractères en ISO15693)</td></tr>
<tr><td>Commentaires</td><td>Les données du bloc mémoire sont des valeurs hexadécimales codées en ASCII (exemple : la valeur 0xAA est représentée par “AA”).</td></tr>
<tr><td rowspan="3">write</td><td>Question</td><td>X | Y | N | D
X : coordonnée en X sur 1 caractère
Y : coordonnée en Y sur 1 caractère
N : numéro du bloc sur 2 caractères
D : données du bloc mémoire (8 caractères en ISO15693)</td></tr>
<tr><td>Réponse</td><td>X | Y | N
X : coordonnée en X sur 1 caractère
Y : coordonnée en Y sur 1 caractère
N : numéro du bloc sur 2 caractères</td></tr>
<tr><td>Commentaires</td><td>Le succès (ou l'échec) de l'écriture est donné par [ACK] (ou [SYN]).</td></tr>
</table>

Remarque :

1. “sid”

La longueur de la trame peut être problématique lorsque de nombreux tags sont détectés par plusieurs antennes. En ISO15693, le SID fait 16 octets (en ASCII).
Si le tag est détecté par 4 antennes, l'ensemble SID + coordonnées a une longueur de 16 + 4x4 = 32 octets. Si 64 tags sont détectés, la longueur totale est de :
(32 + 1) * 64 -1 = 2111 octets, ce qui est supérieur au MTU de l'Ethernet...
Cependant, l'anticolision n'est pas encore opérationnelle.

2. “read” et “write”

L'accès aux tags (pour lire ou écrire dans leur mémoire) se fait pour le moment en sélectionnant l'antenne sur laquelle ils sont. Ceci implique qu'un seul tag doit être vu par l'antenne. Une approche différente devra être envisagée lorsque plusieurs tags pourront être vus par la même antenne (pour le moment, l'anticolision n'est pas traitée).
En ISO15693, les blocs mémoire ont une taille de 4 octets (32 bits) : les données D comportent donc 8 octets.

2.4. Commandes de périphériques (LED et touches tactiles)

Liste des commandes et du corps correspondant (LED) :

<table>
<tr><th>Commande</th><th colspan="2">Format du corps</th></tr>
<tr><td rowspan="3">ledbits</td><td>Question</td><td>4 : 4 bits (N = 1 caractère ASCII) (défaut)
8 : 8 bits (N = 2 caractères ASCII)</td></tr>
<tr><td>Réponse</td><td>idem</td></tr>
<tr><td>Commentaires</td><td>Permet de changer la précision des couleurs (16 millions ou 4096 couleurs par pixel). Le principal intérêt est de diviser par 2 la longueur de la commande leddata, d'autant plus qu'il n'est pas utile d'avoir 16 millions de couleurs, et que la commande leddata devient très longue depuis la version 7.</td></tr>
<tr><td rowspan="3">ledrgb</td><td>Question</td><td><table><tr><td>R</td><td>G</td><td>B</td></tr></table>R : couleur rouge, N caractères (hexadécimal, MSB en premier)
G : couleur verte, N caractères (hexadécimal, MSB en premier)
B : couleur bleue, N caractères (hexadécimal, MSB en premier)</td></tr>
<tr><td>Réponse</td><td>(vide)</td></tr>
<tr><td>Commentaires</td><td>Affecte la couleur RGB spécifiée à toutes les antennes de la dalle.</td></tr>
<tr><td rowspan="3">ledpointrgb</td><td>Question</td><td><table><tr><td>R</td><td>G</td><td>B</td><td>.</td><td>X1</td><td>Y1</td><td>.</td><td>X2</td><td>Y2</td><td>...</td></tr></table>R : couleur rouge, N caractères (hexadécimal, MSB en premier)
G : couleur verte, N caractères (hexadécimal, MSB en premier)
B : couleur bleue, N caractères (hexadécimal, MSB en premier)
X : coordonnée X, 2 caractères (de “01” à “16”)
Y : coordonnée Y, 2 caractères (de “01” à “16”)</td></tr>
<tr><td>Réponse</td><td>(vide)</td></tr>
<tr><td>Commentaires</td><td>Affecte la couleur RGB spécifiée aux LED spécifiées, sans changer les couleurs des autres LED.</td></tr>
<tr><td>leddata</td><td>Question</td><td><table><tr><td>R1</td><td>G1</td><td>B1</td><td>R2</td><td>G2</td><td>B2</td><td>...</td></tr></table>La couleur de chaque antenne est décrite par 3 x N caractères (N par couleur). Voici l'ordre de balayage des antennes :<table><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>17</td><td>18</td><td>19</td><td>20</td></tr><tr><td>5</td><td>6</td><td>7</td><td>8</td><td>21</td><td>22</td><td>23</td><td>24</td></tr><tr><td>9</td><td>10</td><td>11</td><td>12</td><td>25</td><td>26</td><td>27</td><td>28</td></tr><tr><td>13</td><td>14</td><td>15</td><td>16</td><td>29</td><td>30</td><td>31</td><td>32</td></tr><tr><td>33</td><td>34</td><td>35</td><td>36</td><td>49</td><td>50</td><td>51</td><td>52</td></tr><tr><td>37</td><td>38</td><td>39</td><td>40</td><td>53</td><td>54</td><td>55</td><td>56</td></tr><tr><td>41</td><td>42</td><td>43</td><td>44</td><td>57</td><td>58</td><td>59</td><td>60</td></tr><tr><td>45</td><td>46</td><td>47</td><td>48</td><td>61</td><td>62</td><td>63</td><td>64</td></tr></table>A partir de la version 7, le nombre de LED passe à 256 (16 x 16). Il y a 4 LED par antenne. L'ordre de balayage se fait ligne par ligne., en commençant par le coin en haut à gauche (768 caractères à 4 bits par couleur).</td></tr>
</table>

	Commentaires	Affecte les couleurs spécifiées aux antennes correspondantes. **Le nombre de bits par couleur doit être de 4, sinon la trame dépasse le MTU du réseau Ethernet (à partir de la version 7).**
ledrep	Question	0 : réponse aux commandes de led désactivée 1 : réponse aux commandes de led activée (défaut)
	Réponse	idem
	Commentaires	Permet de désactiver la réponse de la dalle aux commandes de led.
ledauto	Question	0 : fonction désactivée (défaut) 1 : fonction activée
	Réponse	idem
	Commentaires	Permet d'ajouter la gestion des led au mode autonome, de la même façon que la commande “sidled”
ledautorgb	Question	R | G | B R : couleur rouge, N caractères (hexadécimal, MSB en premier) G : couleur verte, N caractères (hexadécimal, MSB en premier) B : couleur bleue, N caractères (hexadécimal, MSB en premier)
	Réponse	idem
	Commentaires	Permet de changer la couleur utilisée lors de la commande “sidled” et en mode autonome avec la fonction “ledauto” activée. Par défaut la couleur est : 0, F, 0
leddiviseur	Question	0 : aucune division 1 : division par 2 2 : division par 4 3 : division par 8
	Réponse	idem
	Commentaires	Permet de diminuer l'intensité lumineuse. La valeur est mémorisée en EEPROM avec la commande “progeeprom”.

Remarque :
A partir de la version 7, il y a 4 LED RGB par antenne RFID. Les LED qui sont sur une antenne ont donc des coordonnées différentes de l'antenne. Par exemple, l'antenne RFID de coordonnées (3,2) correspond aux LED de coordonnées (5, 3), (5, 4), (6, 3), (6, 4).

2.5. Commandes de filtre des SID

Liste des commandes et du corps correspondant :

Commande	Format du corps	
bddfiltre	Question	0 : filtre des SID désactivé (défaut) 1 : filtre des SID activé
	Réponse	idem
	Commentaires	Permet d'activer le filtre des SID. Quand le filtre est activé, seuls les SID contenus dans la liste seront envoyés lors des inventaires.
bddajout	Question	SID SID : SID, LSB first (plusieurs octets)
	Réponse	idem
	Commentaires	Permet d'ajouter un SID à la liste des SID mémorisés.
bddclear	Question	(vide)
	Réponse	idem
	Commentaires	Efface la liste des SID mémorisés.

2.6. Commandes de diagnostique

Ces commandes servent à s'assurer du bon fonctionnement de la table.
Liste des commandes et du corps correspondant :

Commande	**Format du corps**	
start	Question	(vide)
	Réponse	(aucune)
	Commentaires	Cette commande est envoyée par la dalle quand elle est prête à fonctionner. L'adresse IP de destination (qui peut être modifiée avec la commande “setserverip”) est 192.168.1.1, sur le port UDP 65000 (qui peut être modifié avec la commande “portUDP”).
erreur	Question	
	Réponse	Numéro | . | Descriptif Numéro de l'erreur sur plusieurs octets.
	Commentaires	
nbcartes	Question	(vide)
	Réponse	N : nombre sur un caractère
	Commentaires	Cette commande lance une détection des cartes RFID, et le nombre N correspond au nombre de cartes RFID détectées. A partir de la version 7, cette commande n'existe plus.
nbantennes	Question	(vide)
	Réponse	N : nombre sur deux caractères
	Commentaires	Cette commande lance une détection des antennes RFID et le nombre N correspond au nombre d'antennes RFID détectées.
version	Question	(vide)
	Réponse	V | . | Date V : version sur 2 caractères Date au format : “YYYYMMDD” (année, mois, jour)
	Commentaires	Cette commande donne le numéro de la version de la dalle et le jour de la programmation.

2.7. Commandes de maintenance

Ces commandes peuvent être utiles pour résoudre des problèmes.
Liste des commandes et du corps correspondant :

Commande	Format du corps	
Xnbant	Question	(vide)
	Réponse	N : nombre sur deux caractères (valeur hexadécimale)
	Commentaires	Le caractère 'X' doit être remplacé par le numéro de la carte RFID ('0' pour toutes les cartes). Cette commande lance une détection des antennes RFID et le nombre N correspond au nombre d'antennes RFID détectées. Cette commande est très rapide (< 10ms) et permet l'observation à l'oscilloscope des signaux du bus SPI des ASIC RFID.
getcfg	Question	(vide)
	Réponse	Chaine de caractères
	Commentaires	Donne la configuration de la dalle (valeurs des paramètres principaux, cartes RFID détectées avec leur nombre d'antennes).
Xgetcfg	Question	(vide)
	Réponse	Chaine de caractères
	Commentaires	Le caractère 'X' doit être remplacé par le numéro de la carte RFID. Donne la configuration de la carte (détail des antennes détectées).
cleareeprom	Question	(vide)
	Réponse	(vide)
	Commentaires	Efface la mémoire EEPROM de la dalle.
readeeprom	Question	N : sur 1 octet, numéro du bloc à lire (de 0 à 4 pour une mémoire de 1K octets).
	Réponse	512 caractères ASCII représentant les données du bloc mémoire.
	Commentaires	Lit la mémoire EEPROM par bloc de 256 octets.
sidled	Question	(vide)
	Réponse	Identique à la commande “sid”
	Commentaires	Commande identique à la commande “sid” à la différence que les antennes qui voient un tag sont éclairées en vert, et les autres sont éteintes.
antled	Question	(vide)
	Réponse	(vide)
	Commentaires	Eclaire en vert les LED qui sont sur les antennes fonctionnelles.
restart	Question	(vide)
	Réponse	aucune
	Commentaires	Redémarre les microcontrôleurs.

3. Modifications depuis la version précédente

- Modification des commentaires de la commande “progeeprom”
- Ajout de la commande “leddiviseur”
- Suppression des commandes de fonctions tactiles

Version précédente : 20091013.