Apprendre la dichotomie avec Colobot

CHABALIER Nicolas

MONCEL Arnaud

Année Universitaire 2014 – 2015

Apprendre la dichotomie avec Colobot

Présenté par CHABALIER Nicolas et MONCEL Arnaud

Tuteur : Jacques LAFFONT

Sommaire

Introduction

Colobot est un logiciel créé pour apprendre à programmer en s'amusant. Le langage enseigné dans ce jeu est un langage qui se rapproche du C++, un langage orienté objet.

Le but de ce projet était, dans un premier temps, de réaliser un scénario composé de plusieurs missions pour faire apprendre de manière ludique la dichotomie en programmation.

Pour se faire nous avons introduit des classes pour faciliter l'apprentissage de la dichotomie et aussi pour faire apprendre les bases de la programmation orientée objet aux joueurs.

La dichotomie étant un processus d'optimisation, nous avons créé une mission dédiée spécialement à l'optimisation avant d'introduire la dichotomie. Dans cette mission l'optimisation se fait sur le sur temps de trajet.

Enfin nous avons introduit la dichotomie dans la 3eme mission en se servant des éléments appris par le joueur dans les 2 premières missions. La difficulté y est donc progressive, tout en rendant l'apprentissage le plus simple et amusant possible.

Cependant nous allons voir que travailler sur un projet déjà existant et l'aspect pédagogique de ce projet ne sont pas choses aisées.

Ce projet a mis à rude épreuve notre créativité quant à l’élaboration de scenario utilisant la dichotomie.

Déroulement du projet

Après avoir pris connaissance de notre projet nous avons tout d'abord pris en main le jeu en faisant toutes les missions exercices qui sont proposées dans le jeu de départ.

Nous avons par la suite téléchargé les sources sur le github qui nous avait été indiqué. Après deux journées à tenter de compiler le code fourni nous avons donc abandonné et travaillé directement sur le jeu qui est aussi disponible sur le github.

Il n'est pas toujours facile de rajouter du contenu à des projets ou jeux déjà crées. Mais les développeurs de Colobot ont prévu qu'une communauté de joueur voudrait rajouter du contenu pour parfaire à leurs attentes.

Nous avons donc trouvé sur le site officiel de Colobot un guide nous permettant de créer nos propres niveaux, cependant ce guide a été conçu pour la version payante de Colobot et non sur la version sur laquelle nous travaillions. Après de longues incompréhensions sur le fait que nos niveaux n'apparaissaient pas dans le jeu nous avons donc décidé de fouiller dans les fichiers du jeu et nous avons trouvé par nous même où placer nos niveaux (data/levels/exercises).

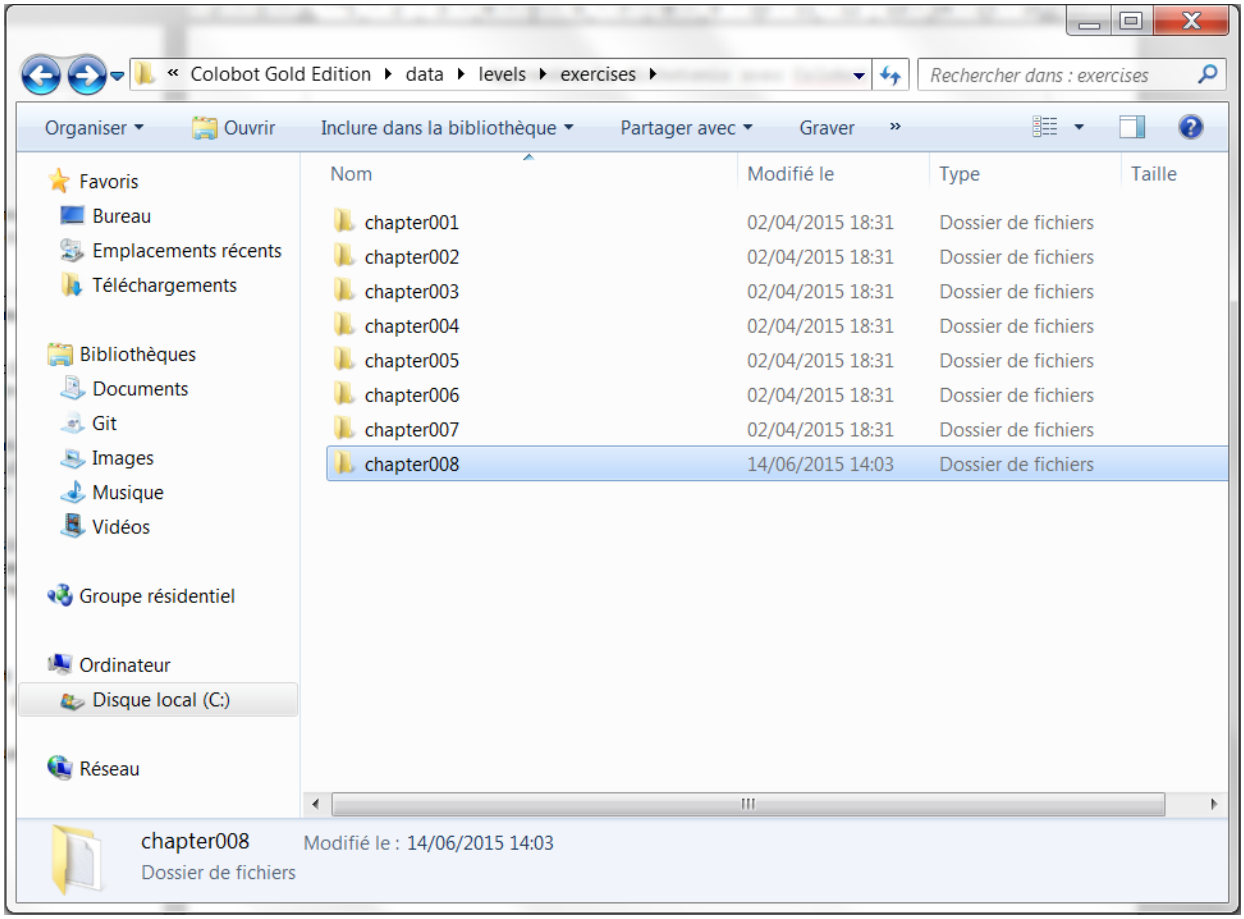
Une fois un premier niveau mis en place il a fallu découvrir par nous-même en parallèle avec le guide comment était structuré un niveau . Les exercices de programmation sont séparés en plusieurs chapitres, nous avons créé le nôtre pour ce projet, il porte le nom de « projet ISIMA ». L'architecture d'un chapitre et de ses niveaux sera expliquée plus bas.

Nous allons maintenant expliquer le travail réalisé, l'architecture des fichiers, les niveaux créés et leurs spécificités.

L'architecture des fichiers et des niveaux

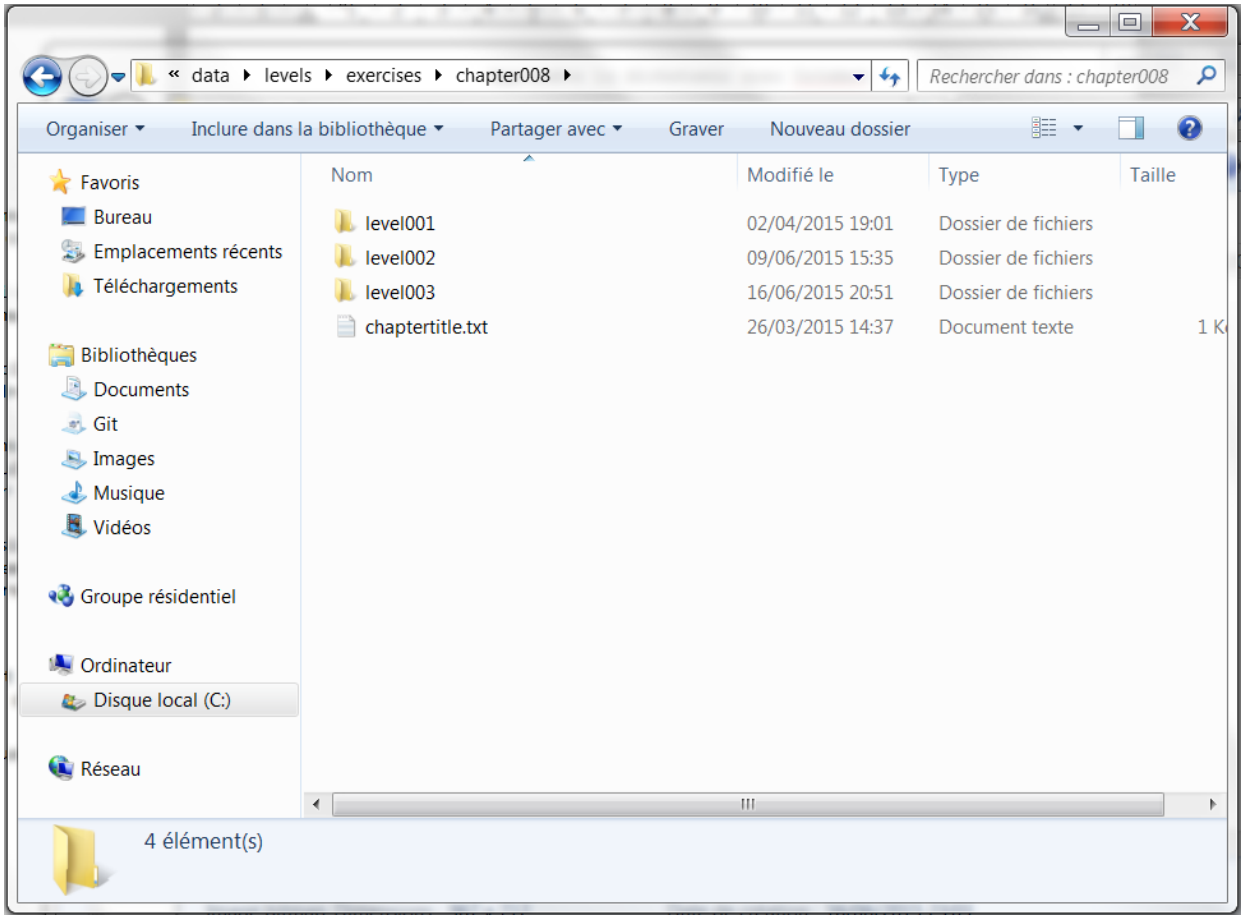
Notre projet se résume en quelque sorte a un chapitre du jeu que nous avons créé pour amener le joueur, dans le dernier niveau, à apprendre de manière ludique le principe de la dichotomie.

Un chapitre est composé de un ou plusieurs niveaux. Dans le point de vue des fichiers sur l'ordinateur un chapitre se compose comme



ceci :

Ici nous avons tous les chapitres correspondant aux exercices de programmation que propose Colobot, notre projet est le chapitre008.



A l'intérieur du chapitre nous pouvons trouver nos niveaux et un fichier donnant des indications sur notre chapitre.

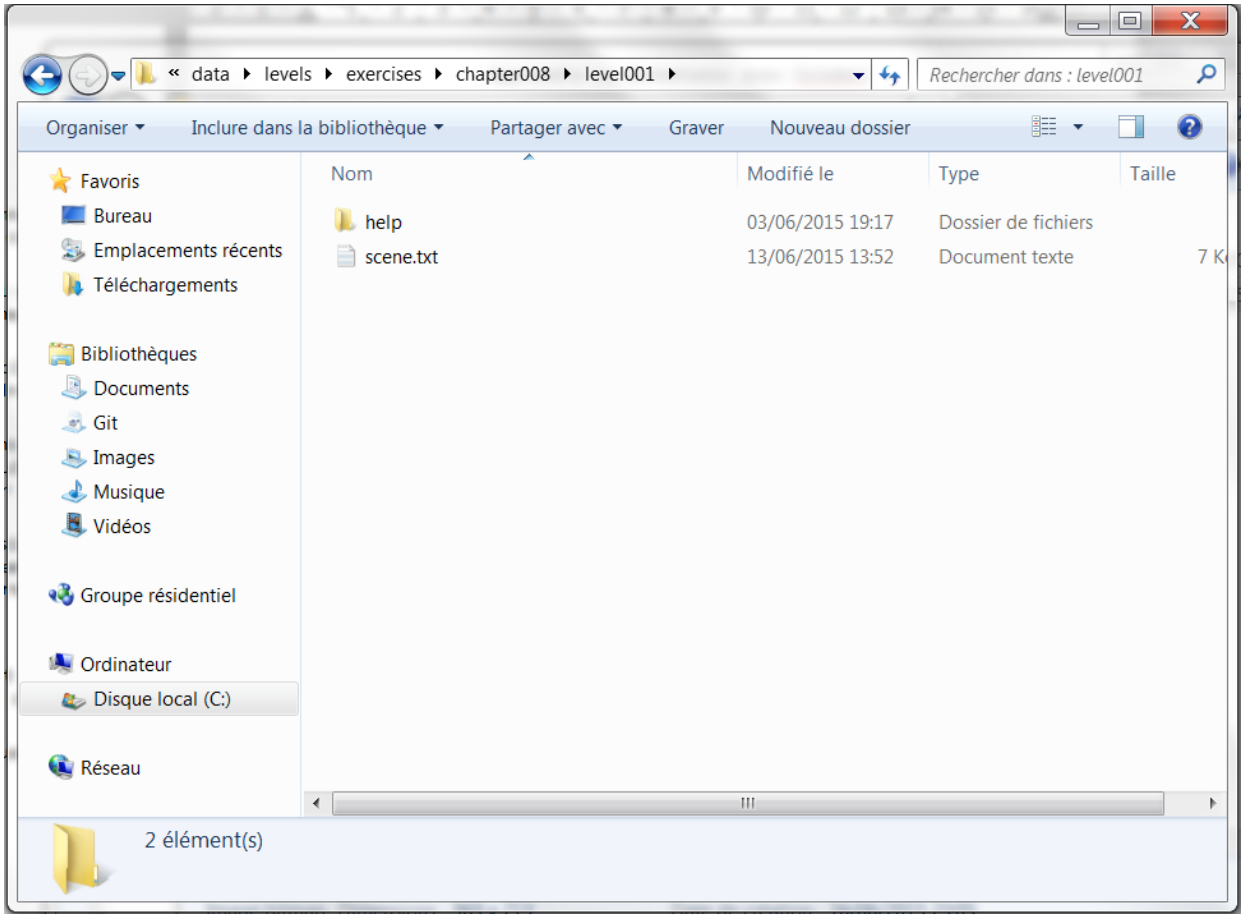
```
1  Title.F text="Projet Isima" resume="Aprendre la dichotomie"
2  // End of level headers translations
```

Voici le contenu du fichier chaptertitle.txt

Dans tous les fichiers de configuration de Colobot le choix de la langue est présent. Pour ce projet nous avons choisi de ne pas traduire tous nos textes en Anglais ou autre langue.

La ligne Title.F définie les informations française du chapitre ici le « text » est pour le nom du chapitre et « resume » est pour la description du chapitre.

Regardons à présent l'architecture des niveaux qui sont ici traduit par level001, level002 etc.



Les niveaux ont eux aussi leurs fichiers de configuration. Le fichier scene.txt sert à définir le nom du niveau, sa description et tout l'ensemble de la scène du niveau. Comme par exemple les conditions de victoire, la forme du terrain, la création des objets déjà présents sur la scène etc.

Le dossier « help » quant à lui contient le fichier d'aide du niveau traduit en toutes les langues. Le fichier d'aide est celui qui est utilisé pour simuler le « scan » dans le jeu ouvert avec la touche « F1 ».

Apprendre la dichotomie avec Colobot

Voilà le résultat en jeu de notre dossier chapitre008 et de ses niveaux.

Détaillons a présent les fichiers help.F.txt et scene.txt

Détail du fichier help.F.txt

Comme dit précédemment le fichier d'aide est retranscris dans le jeu par le biais du « scan ». Il n'est pas difficile à comprendre et à prendre en main.

Très peu d'instruction sont possible il est possible de faire du formatage de texte pour qu'il soit affiché dans le « scan ».

```
1   \b;Houston a un problème !
```

Cette instruction va écrire dans le « scan » « Houston a un problème ! » comme un titre.

```
20  \c;\s;void factoriel(int n)
21  \s;{
22  \s;    if (n<=1)
23  \s;        return n;
24  \s;    else
25  \s;        return n * factoriel(n-1);
26  \s;}
27  \n;
```

Ici « \c ; » déclare une section de code et va afficher dans le « scan » le texte qui se trouve sous la balise « \s ; » dans un fond jaune. Le « \n ; » traduit un saut de ligne.

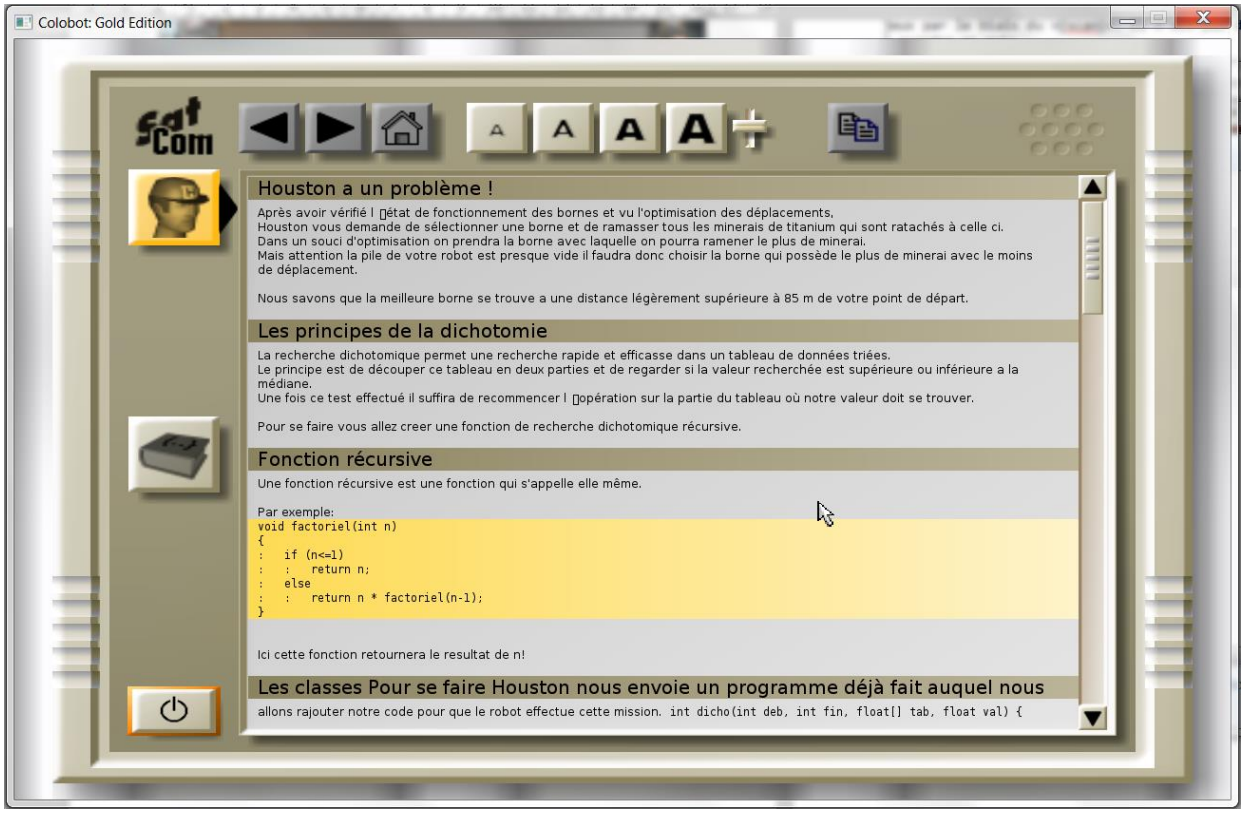
```
33  \image isiClasse 30 15;
```

Cette instruction affichera une image nommée isiClasse dans le « scan » le 30 et 15 sont pour la taille de l'image. Mais attention l'image devra être dans le bon dossier (data/icons).

Apprendre la dichotomie avec Colobot

```
\c;\l;distance\u cbot\dist;\n;
```

Cette instruction permet de créer un lien vers une fonction de Colobot existante dans le guide de programmation en jeu. Le « \l ; » déclare un lien nommé distance vers une fonction de Colobot qui est « dist ».



Exemple d'aide en jeu :

Détail du fichier scene.txt

Le fichier scene.txt quant à lui défini tout le niveau et ses conditions de jeux.

```
1  Title.F text="La dichotomie"
2  Resume.F text="Apprendre la dichotomie."
3  ScriptName.F text="finder"
4  // End of level headers translations
```

On connaît déjà les deux premières lignes elles sont similaires à celles du chapitre. La ligne importante est la troisième, cette ligne signifie que lorsque vous créez un nouveau script sur un des robots il s'appellera dans notre cas « finder ».

```
6  Instructions name="%lvl%/help/help.%lng%.txt"
7  HelpFile name="cbot.txt"
```

La ligne « instructions name » lit le fichier d'aide vu auparavant. La Ligne d'après et très importante c'est elle qui lit l'aide principale du jeu.

```
11  AmbientColor air=0.400;0.400;0.400;0.400 water=0.078;0.078;0.078;0.078 // grey
12  FogColor air=0.306;0.306;0.498;0.000 water=0.263;0.314;0.392;0.000 // magenta
13  VehicleColor color=0.659;0.620;0.463;0.000 // sable
14  DeepView air=125.00 water=25.00
15  FogStart air=0.9 water=0.5
16  SecondTexture rank=1
17  ForegroundName image="lens5.png"
18  Background up=0.753;0.627;0.627;0.000 down=0.816;0.753;0.502;0.000 cloudUp=0.502;0.
19
20  TerrainGenerate vision=250.00 depth=1 slope=3.0
21  TerrainWind speed=3.0;0.0
22  TerrainRelief image="relief55.png" factor=1.00
23  TerrainResource image="res00.png"
24  TerrainCloud image="cloud09.png" level=125.0
25
26  TerrainMaterial id=1 image="desert4" u=0.00 v=0.00 up=1 down=1 left=1 right=1 hard=0.2
27  TerrainMaterial image="desert4" u=0.25 v=0.00 up=2 down=1 left=1 right=1 hard=0.4
28  TerrainMaterial image="desert4" u=0.50 v=0.00 up=1 down=1 left=1 right=2 hard=0.4
29  TerrainMaterial image="desert4" u=0.75 v=0.00 up=2 down=1 left=1 right=2 hard=0.4
30  TerrainMaterial image="desert4" u=0.00 v=0.25 up=1 down=2 left=1 right=1 hard=0.4
31  TerrainMaterial image="desert4" u=0.25 v=0.25 up=2 down=2 left=1 right=1 hard=0.4
32  TerrainMaterial image="desert4" u=0.50 v=0.25 up=1 down=2 left=1 right=2 hard=0.4
33  TerrainMaterial image="desert4" u=0.75 v=0.25 up=2 down=2 left=1 right=2 hard=0.4
34  TerrainMaterial image="desert4" u=0.00 v=0.50 up=1 down=1 left=2 right=1 hard=0.4
35  TerrainMaterial image="desert4" u=0.25 v=0.50 up=2 down=1 left=2 right=1 hard=0.4
36  TerrainMaterial image="desert4" u=0.50 v=0.50 up=1 down=1 left=2 right=2 hard=0.4
37  TerrainMaterial image="desert4" u=0.75 v=0.50 up=2 down=1 left=2 right=2 hard=0.4
38  TerrainMaterial image="desert4" u=0.00 v=0.75 up=1 down=2 left=2 right=1 hard=0.4
39  TerrainMaterial image="desert4" u=0.25 v=0.75 up=2 down=2 left=2 right=1 hard=0.4
40  TerrainMaterial image="desert4" u=0.50 v=0.75 up=1 down=2 left=2 right=2 hard=0.4
41  TerrainMaterial id=2 image="desert4" u=0.75 v=0.75 up=2 down=2 left=2 right=2 hard=0.6
```

Toutes ces lignes sont pour la création du terrain. Les premieress sont pour les lumières et couleurs ambiantes.

Le deuxième paragraphe est pour la fonte du terrain comme le relief créé à partir d'une normal map.

Enfin les dernières sont là pour appliquer une texture au terrain.

```
66  BeginObject
67  CreateObject pos=00.00;00.00 dir=1.2 type=Me option=1
68  CreateObject pos=00.00;10.00 dir=1.0 type=LeggedGrabber reset=1 trainer=1 script1="initPI3.txt" power=0.2
69  CreateObject pos=00.00;10.00 dir=1.0 type=StartArea
70
71
72  //les balise avec leur rafinerie et leur minerais
73  CreateObject pos=-50.00;00.00 dir=0.0 type=ExchangePost
74  CreateObject pos=-40.00;-04.00 dir=0.0 type=Converter
75  CreateObject pos=-45.00;05.00 dir=0.0 type=GoalArea
76  CreateObject pos=-100.00;10.00 dir=0.0 type=TitaniumOre
77  CreateObject pos=-10.00;01.00 dir=0.0 type=TitaniumOre
78  CreateObject pos=-20.00;-15.00 dir=0.0 type=TitaniumOre
79  CreateObject pos=-110.00;00.00 dir=0.0 type=TitaniumOre
```

Les lignes importantes sont celles-ci, c'est ici que les objets du niveau se créés. Les instructions sont les mêmes pour tous les objets. On définit l'objet qu'on veut créer par son type. Il en existe une multitude et heureusement pour nous ils étaient tous dans le guide.

Attardons nous sur la ligne « 68 » du code ci-dessus, la position de l'objet est donnée par « pos », sa direction par « dir », son type ici « leggedGrabber » est un robot qui peut attraper des objets. L'option « reset=1 » signifie que l'objet est réinitialisé à chaque nouvelle partie. L'option « trainer=1 » signifie que c'est un robot d'entraînement et qu'il est le seul à pouvoir déclencher les points de passage « wayPoint » dans le jeu. L'option « power=0,2 » est l'état de charge de sa pile au commencement du jeu, cette option peut être réglée de 0 à 100. De 0 a 1 pour la pile normale et de 1 à 100 pour la pile d'uranium.

L'instruction « script1=« initPI3.txt » » quant à elle permet de lier un script fait par les développeurs au robot dès le début. Accessoirement il est utilisé pour les scripts solutions. Attention toutefois le script « initPI3.txt » doit se trouver au bon endroit (data/ai).

```
89  extern void object::Finder( )
90  {
91      //code
92  }
```

Il est très important que le script possède le nom donné plus haut ici « Finder » sinon il ne sera pas pris en compte.

Problèmes rencontrés et solutions adoptées

Les problèmes ont été multiples, tout d'abord les problèmes de compilations suite à une erreur de syntaxe d'une librairie externe. Que nous avons laissé de côté pour nous concentrer sur le projet et la création de niveau introduisant la dichotomie.

La recherche de scenario n'a pas été une mince affaire, surtout pour faire en sorte d'utiliser la dichotomie dans l'un d'eux. Suite à de long moment de réflexions nous avons décidé de scinder en plusieurs niveaux notre projet pour introduire doucement la dichotomie, et donner un scenario qui soit jouable et ayant un minimum d’intérêt.

Nous avons commencé par introduire une notion de classe dans le niveau 1. Et ensuite une notion d'optimisation dans le niveau 2 et pour finir la dichotomie dans le niveau 3.

Nous avions en têtes nos niveaux, mais par où commencer ? Comment crée un niveau ? De longues heures durant nous avons cherché, suite au non fonctionnement des instructions du guide, où placer nos niveaux et notre code.

Après avoir trouvé, nous avons eu l’idée de se servir des fonctionnalités du jeu et de faire découvrir à l'utilisateur comment lire un fichier en c++, ce qui allait donner à notre scenario un meilleur déroulement. Le problème c'est que cette fonctionnalité dans le jeu gratuit n'a pas était bien codée et l'utilisation de celle-ci provoque un crash du jeu.

Nous avons donc abandonné cette idée la et codé en dur des valeurs permettant le bon déroulement de notre scenario (c'est à dire les insérer directement dans le code de la fonction).

Nous allons par la suite présenter le scenario dans ces grandes lignes et les classes créées pour pouvoir le mener à bien.

Description du scenario et des classes

Scenario

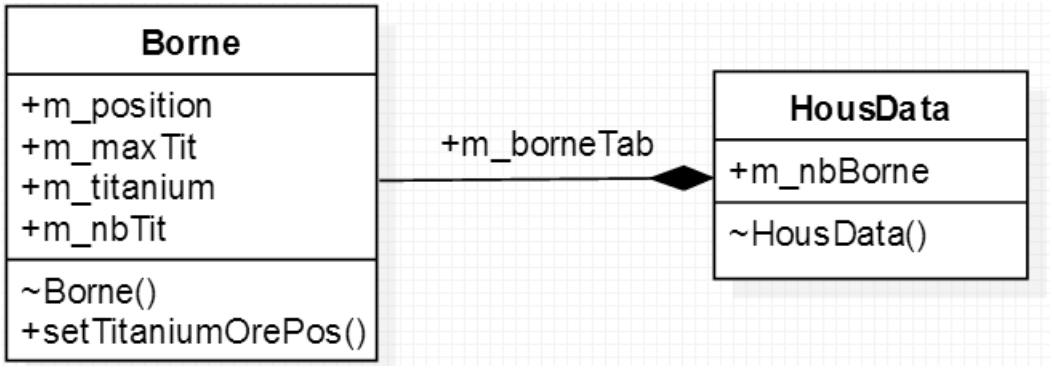
Pour pouvoir utiliser la dichotomie dans un scenario il fallait avoir des tableaux et jouer sur de l'optimisation. Nous avons choisi d'effectuer des missions sur la récupération de minerai de titanium un peu partout sur la carte. Le scenario réside sur l'envoie de donné (fictive) par Houston (dirigeant de la mission spécial) à notre astronaute pour récupérer le maximum de titanium avant la fin de vie du robot symbolisé par la charge de sa pile.

Le principe est d'avoir plusieurs Borne sur la carte, qui ont un tableau en donnée membre contenant la position de minerai de titanium dans ses alentours. Toutes ces bornes sont rassemblées dans une même classe pour ne former qu'un seul objet facile d'utilisation.

Nous avons donc créé un système de classe qui nous permettait d'apprendre l'utilisation de données membres de classe. Les informations envoyées par Houston sont symbolisées par notre code qui va vous être présenté.

Classes

Voici le schéma de classe que nous avons créé :



Une classe borne ayant une position (m_position), un nombre de minerais de titanium(m_nbTit), et un tableau de position de ces minerais(m_titanium) comme données membres.

Elle a aussi une fonction setTitaniumOrePos(), cette fonction n'est pas destinée au joueur dû au fait qu'on ne puisse pas lire un fichier depuis le jeu. C'est écrit en « dur » et les positions sont directement données.

La classe HousData rassemble les bornes, elle possède en données membres le nombre de Borne qu'elle possède (m_nbBorne) et un tableau de Borne correspondant à la classe définie précédemment.

Rapide présentation du code

```
76  extern void object::Finder( )
77  {
78      HousData houstonInfo();
79
80      //ici le code a placer
81  }
```

Dans toutes les missions c'est cet objet houstonInfo que nous avons définie comme étant les données envoyées par Houston. C'est dans cet objet que se trouve le tableau de Bornes.

```
1   //fonction qui decrit une borne
2   public class Borne
3   {
4       //position de la borne
5       point m_position;
6
7       //nombre max de minerais pour la borne
8       int m_maxTit;
9
10      //tableau de mierais
11      point m_titanium[5];
12
13      //nombre de minerais dans le tableau
14      int m_nbTit = 0;
15
16      //constructeur de la classe
17      void Borne(int x, int y, int maxTit)
18      {
19          int z = topo(x, y);
20          m_position = new point(x, y, z);
21          m_maxTit = maxTit;
22      }
23
24      //fonction qui ajoute un minerai au tableau
25      void setTitaniumOrePos(int x, int y)
26      {
27          if(m_nbTit < m_maxTit)
28          {
29              int z = topo(x, y);
30              m_titanium[m_nbTit] = new point(x, y, z);
31              m_nbTit++;
32          }
33      }
34  }
```

La classe Borne qui lors de sa création demande le nombre de titanium qu'elle possède et sa position en paramètre.

```
36  //fonction qui vas rassembler toute les donees de chaque borne
37  public class HousData
38  {
39      //nombre de borne
40      int m_nbBorne = 4;
41
42      //tableau de borne
43      Borne m_borneTab[4];
44
45      //initialisation en "dur" malheureusement du au crash du jeu quand on tente de lire un fichier
46      void HousData()
47      {
48          //creation de la borne 1 ainsi que ces minerais
49          m_borneTab[0]=new Borne(-45.00, -02.00, 4);
50          m_borneTab[0].setTitaniumOrePos(-100.00, 10.00);
51          m_borneTab[0].setTitaniumOrePos(-10.00, 01.00);
52          m_borneTab[0].setTitaniumOrePos(-20.00, -15.00);
53          m_borneTab[0].setTitaniumOrePos(-110.00, 00.00);
54
55          //creation de la borne 2 ainsi que ces minerais
56          m_borneTab[1]=new Borne(-15.00, 78.50, 4);
57          m_borneTab[1].setTitaniumOrePos(-20.00, 65.00);
58          m_borneTab[1].setTitaniumOrePos(-10.00, 115.00);
59          m_borneTab[1].setTitaniumOrePos(-60.00, 75.00);
60          m_borneTab[1].setTitaniumOrePos(30.00, 105.00);
61
62          //creation de la borne 3 ainsi que ces minerais
63          m_borneTab[2]=new Borne(85.00, -15.00, 1);
64          m_borneTab[2].setTitaniumOrePos(100.00, 00.00);
65
66          //creation de la borne 4 ainsi que ces minerais
67          m_borneTab[3]=new Borne(200.00, 15.00, 3);
68          m_borneTab[3].setTitaniumOrePos(210.00, 10.00);
69          m_borneTab[3].setTitaniumOrePos(200.00, 60.00);
70          m_borneTab[3].setTitaniumOrePos(250.00, 50.00);
71
72
73      }
74  }
```

La classe HousData avec dans son constructeur la création des Bornes et la mise à jour du tableau de position des minerais pour chacune d'elles.

Description des niveaux

Dans tous les niveaux, la solution est écrite dans l'aide car il nous était impossible de rajouter un deuxième script sur le robot lors de sa création.

Niveau 1

Ce niveau amène le joueur à prendre connaissance des classes et de leur utilisation. Il lui sera demandé à l'aide du robot de parcourir le tableau de Borne et de rendre visite à chacune d'elles.

La victoire est acquise quand le robot a visité toutes les Bornes et est passé sur tous les « wayPoint ». La pile du robot est à un niveau de batterie tel que seul le programme juste peu réussir.

Niveau 2

Dans ce niveau, le but est de faire de l'optimisation de trajet. Nous avons accès à une borne et ses minerais. Le but est de ramener le plus de minerai possible sur les plateformes prévus pour cela.

Sur la carte, il y a 4 minerais disposés mais la pile du robot ne peut tenir que si le robot prend les 3 minerais les plus proches.

Il faut donc stocker les coordonnées des minerais dans un tableau trié par distance entre le minerai et la plateforme et demander au robot d'aller chercher les minerais en commençant pas les plus proches.

Niveau 3

Ce niveau quant à lui va amener le joueur à se servir de la dichotomie pour aller récupérer les minerais se trouvant autour d'une borne. La Borne est choisie par Houston et le joueur devra la rechercher avec la dichotomie suivant leurs distances par rapport à lui.

Le niveau est gagné quand tous les minerais de titanium de la borne choisie sont ramassés.

Conclusion

Pour conclure, ce projet nous a permis de développer nos connaissances en C ainsi qu'en C++. Nous avons aussi appris à comprendre un programme que nous ne connaissions pas pour pouvoir l'adapter et créer de nouvelles missions.

Travailler sur un projet réalisé par d’autre et qui nous est inconnu n'est pas facile, mais nous sommes fiers d'avoir mené ce projet à son terme.

Cela nous a aussi appris les bases en matière de jeux en 3D pour gérer les éléments et aussi pour créer une carte. Il nous a fait comprendre que trouver des exercices pour faire apprendre de manière ludique n'est pas toujours simple.

La leçon que nous retiendrons est qu'il est très important, si notre projet risque d’être repris ou d'être analysé par une communauté, que le code soit commenté et que le projet soit fourni d'un guide d'explication clair (comme ici pour la création de niveaux).

Nous avons aussi demandé à un de nos amis de tester nos 3 niveaux alors qu'il ne comprenait pas comment marchait la dichotomie. Et il a admis qu'il comprenait mieux maintenant grâce à nos exercices.

Nous tenions a remercier ISIMA et plus particulièrement Mr Jacques LAFFONT notre tuteur pour nous avoir donné ce projet.