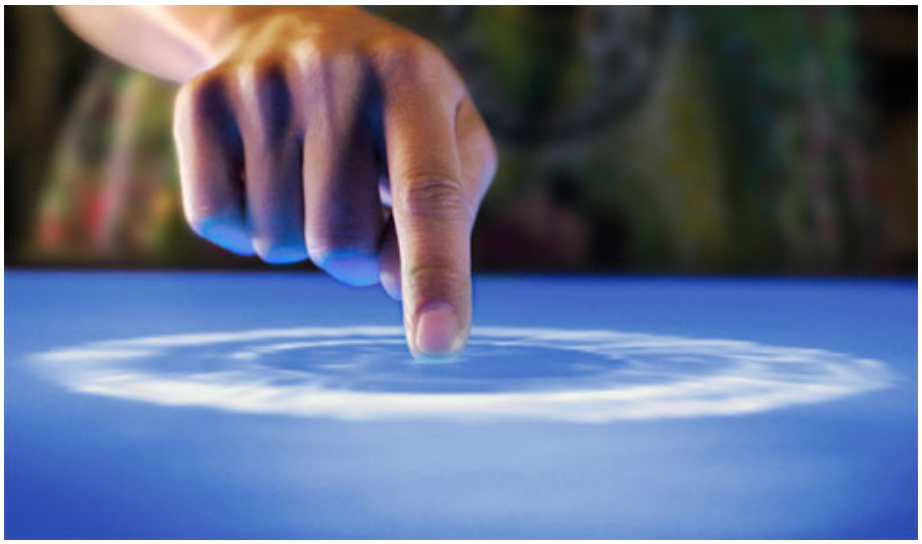
Rapport final de stage

ScatterTangram



Réalisé par

Damien Sendner

Sous la direction de

Lorna Mcknight, Brendan Cassidy et Serge-André Mahé

Année universitaire 2009-2010

Avant-propos

Ce projet a été mené en collaboration avec mon binôme Thibaud Belin, j'emploierai donc parfois le « nous » plutôt que le « je » au cours de ce rapport étant donné que nous ne nous sommes pas clairement attribuées les tâches.

Cependant, on peut dire que Thibaud s'occupait plutôt de la partie ergonomie et utilisateur tandis que je m'occupais des côtés plus techniques.

Remerciements

Je tiens à remercier tout d'abord nos tuteurs de l'UCLAN, Brendan Cassidy et Lorna Mcknight qui nous ont guidé tout au long de ce projet en nous fournissant de l'aide et des pistes de recherches notamment sur la détection de collision, l'ergonomie et le développement sur Microsoft Surface.

Je tiens de plus à remercier, notre tuteur en France, Serge-André Mahé qui nous a donné de nombreux conseils sur la poursuite de notre projet durant sa visite à mi-stage.

Un grand merci à Janet Read pour son accueil au début du stage ainsi que pour son cours très instructif sur le design participatif. Merci à Daniel Fitton pour son aide dans l'utilisation de la table Surface. Enfin, merci à toute l'équipe du Chici lab pour le matériel qu'ils ont mis à notre disposition.

Table des matières

Glossaire **5**

Introduction **7**

1 L'UCLAN et le Chici Lab **8**

2 Cahier des charges **9**
2.1 Le jeu . . . 9
2.1.1 Qu'est ce que le jeu du tangram ? . . . 9
2.1.2 Comment doit-être ce jeu ? Que doit-il implémenter ? . . . 9
2.2 La plateforme . . . 10
2.3 Les cas d'utilisations . . . 10

3 Outils et langages **11**
3.1 .NET et C♯ . . . 11
3.1.1 Qu'est ce que le Microsoft .NET ? . . . 11
3.1.2 Le langage C♯ . . . 12
3.2 WPF et Xaml . . . 12
3.2.1 Qu'est ce que XAML ? . . . 12
3.2.2 Qu'est ce que XAML apporte de plus dans la création d'interface graphique ? . . . 12
3.3 Microsoft Surface . . . 13
3.3.1 La table tactile . . . 13
3.3.2 Microsoft Surface . . . 14
3.4 Autres outils . . . 14
3.4.1 Microsoft Office OneNote 2007 . . . 14
3.4.2 Microsoft Office Visio 2007 . . . 14

4 Ergonomie **15**
4.1 Interface sous Microsoft Surface . . . 15
4.1.1 Interface n° 1 . . . 15
4.1.2 Interface n° 2 . . . 15
4.1.3 Choix . . . 16
4.2 Design participatif . . . 16
4.3 Autres aspects ergonomiques . . . 16

5 Analyse et Programmation **18**
5.1 Vue d'ensemble . . . 18
5.2 ScatterView . . . 19
5.3 Interface avec WPF . . . 19
5.4 Le tactile . . . 21
5.5 Le mouvement : *ContactChanged* . . . 21

5.6 Gestion des données 21
5.6.1 Quelles données ? 21
5.6.2 Sauvegarde des données (du .txt au .xml) 22
5.6.2.1 Dans un fichier texte 22
5.6.2.2 Dans un fichier XML 22
5.6.3 Traitement des données(*ShapesManager*) 23
5.6.3.1 Comment récupère-t-on les données d'un fichier texte ? 23
5.6.3.2 Comment récupère-t-on les données d'un fichier XML ? 24
5.6.3.3 La conversion des données en TangramShape 24
5.6.3.4 La miniaturisation des formes pour le menu (*minimizeShape*) . 25
5.7 Moteur(*TangramShape*) 25
5.8 L'enregistrement des scores 26
5.9 Detection de collision 26
5.9.1 Faire un détecteur de collision 26
5.9.2 XNA 27
5.9.3 Surface Physics 27

6 Discussion **28**
6.1 Résultats obtenus et résultats attendus 28
6.2 Optimisations possibles du projet 28

Conclusion **29**

A Le code pour changer le style de la ScatterView **31**

B La classe ShapesManager avec un fichier texte **32**

C La classe ShapesManager avec un fichier XML **39**

Glossaire

SDK (Software Development Kit) : Un kit de développement ou trousse de développement logiciel est un ensemble d'outils permettant aux développeurs de créer des applications de type défini (ex pour iphone ou ipod).

Framework : un framework est un kit de composants logiciels structurels, qui définissent les fondations ainsi que les grandes lignes de l'organisation de tout ou partie d'un logiciel (architecture).

IDE (Integrated Development Environment) : Un environnement de développement intégré est un programme regroupant un ensemble d'outils pour le développement de logiciels. En règle générale, un EDI regroupe un éditeur de texte, un compilateur, des outils automatiques de fabrication, et souvent un débogueur.

Table des figures

2.1 Le Tangram 9
2.2 Diagramme de Cas d'utilisation 10

3.1 Common Language Infrastructure 11
3.2 Structure de la table tactile 13
3.3 Le logo de Microsoft Surface 14

4.1 Interface n° 1 15
4.2 Interface n° 2 17

5.1 Diagramme de classe 18
5.2 Les fichiers XAML 20
5.3 La création d'événement en C♯ 20
5.4 Les fichiers XAML et C♯ de l'interface 21
5.5 Le fichier .txt 22
5.6 Le fichier .xml 23
5.7 Création du Path 25
5.8 Méthode de détection de collision 27

Introduction

Le tactile est une technologie assez vieille par sa création, mais qui commence seulement aujourd'hui à rentrer dans les moeurs. Nous commenons à être envahi par le tactile, tout d'abord dans notre téléphone, puis dans les ordinateurs, dans les lieux publics, caisse automatique, distributeur de billet, etc. Bientôt il va s'inviter dans notre salon avec notamment Microsoft Surface une table de salon tactile.

Mon projet consiste en l'élaboration d'une application nommée « ScatterTangram » pour cette table tactile. Ce jeu est basé sur le jeu chinois du Tangram qui consiste à reconstituer une forme géométrique à partir de plusieurs pièces comme un puzzle.

Je vais commencer par établir les spécifications de ce projet à travers le cahier des charges, puis nous verrons dans un deuxième temps toutes les technologies que j'ai dues apprendre pour mener à bien ce projet. Ensuite nous développerons le côté ergonomie et utilisation de notre logiciel et nous poursuivrons par les aspects techniques de celui-ci. Pour finir, nous discuterons sur les possibles applications de ce projet et surtout sur le futur de la technologie sur lequel il s'appuie.

Chapitre 1

L'UCLAN et le Chici Lab

Je vais tout d'abord vous présentez le contexte dans lequel j'ai effectué ce stage. J'ai travaillé dans l'Université du Central Lancashire (UCLAN) à Preston. Je n'ai donc pas travaillé dans une ambiance d'entreprise, mais plutôt dans une ambiance de recherche. En effet, nous n'avions pas d'horaires, seulement un rendez-vous une fois par semaine avec nos tuteurs pour rendre compte de notre avancement. Les enseignant-chercheurs de l'UCLAN ont mis à notre disposition une salle appelée "ChiCi Lab", qui contenait beaucoup de matériel : des ordinateurs, une télévision, un rétroprojecteur, etc. De plus nous avions accès à une bibliothèque ouverte 24h sur 24.

Malgré le matériel et les locaux mis à notre disposition une des difficultés réside dans le radical changement de travail par rapport à l'IUT et notamment à notre premier projet. Dans celui-ci nous étions guidés dans nos choix, car le projet avait déjà été effectué auparavant. Au cours de ce stage au contraire, le projet n'a jamais été réalisé, de plus je travaillais sur une des dernières technologies tactiles, étant donné son prix peu de monde possède cette table tactile donc par conséquent il y a peu de ressources que j'ai pu exploiter pour trouver des informations. Je me suis servi uniquement des quelques tutoriaux que j'ai pu trouver sur internet et du forum MSDN [6]. Le reste de mon travail a consisté à décortiquer les logiciels déjà présents sur Microsoft Surface pour découvrir comment ils fonctionnaient.

On peut donc dire que notre travail était vraiment différent d'un travail en entreprise dans lequel on reçoit des directives. La majeure partie de celui-ci consisté à s'auto-former sur les technologies liées au développement sur Microsoft Surface.

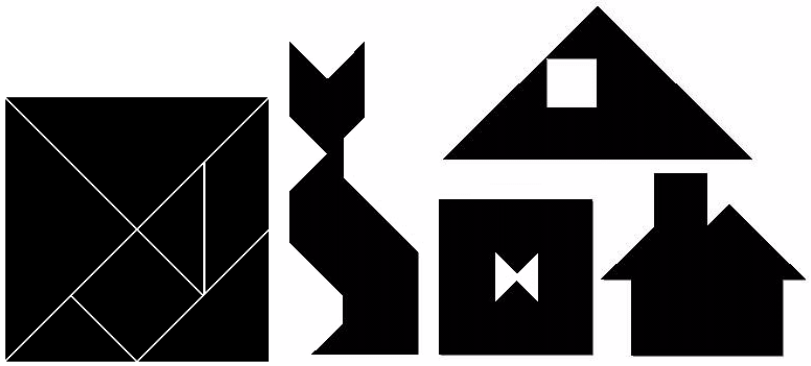
Chapitre 2

Cahier des charges

2.1 Le jeu

2.1.1 Qu'est ce que le jeu du tangram ?

D'après Wikipedia [9], le tangram est un ancien jeu de solitaire chinois. Il se compose de sept pièces pouvant reconstituer un carré (Voir image 2.1(a)). À partir de ces pièces on peut construire un grand nombre de formes géométriques ou figuratives (voir image 2.1(b)). Ce jeu peut être utilisé comme un casse-tête, celui-ci consistant à présenter une forme au joueur qu'il devra reconstituer à partir des sept formes de base.



(a) Le carré (b) Formes constituées à partir du carré

FIGURE 2.1 – Le Tangram

C'est donc ce jeu que nous allons implémenter sur la table tactile équipée de Microsoft Surface. Cependant, nous allons étendre les possibilités de celui-ci en permettant que le nombre et la forme des pièces de base puissent changer d'une forme à l'autre.

2.1.2 Comment doit-être ce jeu ? Que doit-il implémenter ?

- Le joueur doit avoir le choix entre plusieurs formes à reconstituer, quand il en choisit une, celle-ci se divise en plusieurs pièces.
- Les pièces doivent pouvoir être déplacées par le joueur.
- On doit pouvoir effectuer une rotation des pièces.
- Quand une pièce est sur le bon emplacement elle doit se bloquer sur celui-ci. Ce bloquage doit être une option qui peut être désactivée.
- Le joueur doit pouvoir sélectionner les formes suivant leur difficulté.
- Le jeu doit enregistrer le temps que met le joueur pour finir de compléter une forme.

– Le jeu doit posséder un système physique qui permet de simuler des pièces réelles c'est-à-dire qu'elles doivent se pousser entre elles, etc...

2.2 La plateforme

Ce jeu devra être implémenté sur la table tactile Microsoft Surface (Voir chapitre 3.3). Il devra donc s'adapter à son interface et aux possibilités offertes par celle-ci. Les applications pour Microsoft Surface peuvent être uniquement développées, grâce à la plateforme .NET (voir chapitre 3.1.1). On doit utiliser le Microsoft Surface **SDK**, qui permet de faire des applications Surface en C♯ (voir chapitre 3.1.2) avec WPF(voir chapitre 3.2).

2.3 Les cas d'utilisations

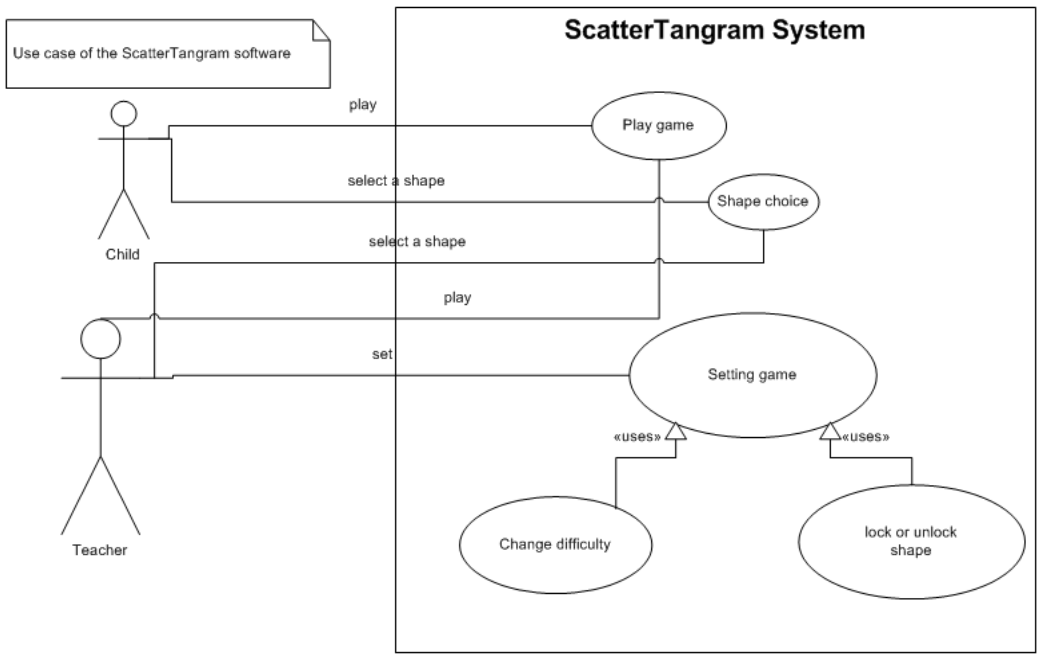


FIGURE 2.2 – Diagramme de Cas d'utilisation

Cette application est faite pour de très jeunes enfants, le scénario de base se passe dans un cadre pédagogique. Il y a donc deux acteurs, le professeur et les enfants, dans un premier temps le professeur ouvre l'application et régle les paramètres c'est à dire notamment la difficulté, ensuite il peut laisser le jeu aux enfants qui peuvent choisir la forme qu'ils veulent puis commencer à jouer. Bien sûr le professeur peut jouer aussi.

Chapitre 3

Outils et langages

3.1 .NET et C♯

3.1.1 Qu'est ce que le Microsoft .NET ?

Selon l'encyclopédie libre Wikipedia [8], C'est un ensemble de technologies Microsoft qui permettent de faciliter la programmation. Il contient notamment le **framework** .NET qui est un ensemble de composants logiciels offrant un grand nombre de possibilités au programmeur. Il possède aussi plusieurs langages tels que le C♯, VB.NET ou J♯, ainsi que l'**IDE** Visual Studio. Enfin il possède une machine virtuelle basée sur la CLI multi-langage, celle-ci permet une interopérabilité entre différents langages c'est-à-dire que quelque soit le langage utilisé (C♯, VB.NET ou J♯) celui-ci lors de la compilation sera transformé dans un langage appelé Common Intermediate Language (CIL) . Ce CIL sera ensuite transformé en langage machine (voir Image 3.1). Pour pouvoir développer sur Microsoft Surface nous allons donc devoir utiliser Visual Studio avec le langage C♯.

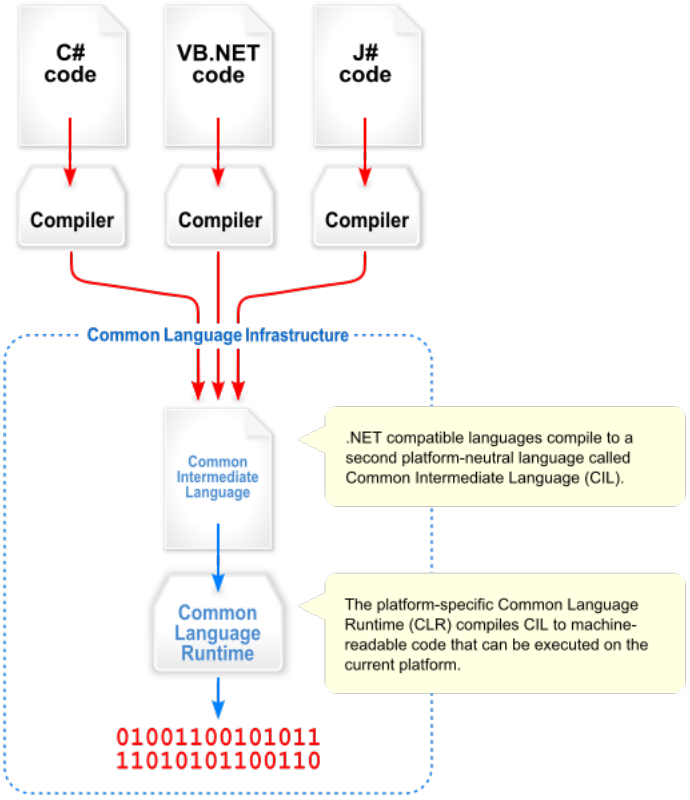


FIGURE 3.1 – Common Language Infrastructure

3.1.2 Le langage C♯

D'après Wikipedia [7] :

> Le C♯ est un langage de programmation orienté objet à typage fort, créé par la société Microsoft, et notamment un de ses employés, Anders Hejlsberg, le créateur du langage Delphi. Il a été créé afin que la plate-forme Microsoft .NET soit dotée d'un langage permettant d'utiliser toutes ses capacités. Il est très proche du Java dont il reprend la syntaxe générale ainsi que les concepts (la syntaxe reste cependant relativement semblable à celles de langages tels que le C++ et le C).

Le java et le C♯ sont donc deux langages très proches de par leur syntaxe mais il y a aussi beaucoup de concepts qui fonctionnent de la même façon. Cependant il existe tout de même quelques différences comme les « delegates » que j'aborderai dans le chapitre 5.3 ou les classes partielles qui sont seulement un moyen de séparer une classe dans deux fichiers différents.

3.2 WPF et Xaml

D'après Wikipedia [10] :

> *Windows Presentation Foundation (WPF)* est la spécification graphique de Microsoft .NET 3.0. Il intègre le langage descriptif XAML qui permet de l'utiliser d'une manière proche d'une page HTML pour les non développeurs.

Pour résumer une application WPF est une application qui utilise du XAML (voir chapitre 3.2.1) pour générer son interface graphique et un autre langage tel que C♯ ou VB.Net pour le moteur. Le moteur est donc complètement séparé de l'interface graphique.

3.2.1 Qu'est ce que XAML ?

D'après Wikipedia [11] :

> XAML est un langage basé sur le langage XML déclaratif développé pour les besoins d'un système d'exploitation de Microsoft, Windows Vista. Prononcé Zammel, ces initiales correspondent à eXtensible Application Markup Language.

3.2.2 Qu'est ce que XAML apporte de plus dans la création d'interface graphique ?

Il permet la création d'interface graphique beaucoup plus simplement. En effet, ce langage est un dérivé du langage XML donc il est stucturé de la même façon, on peut donc imbriquer les différents éléments de notre interface de manière logique. Le mieux est de voir un exemple avec ou sans WPF :

Sans WPF :

```
public partial class Window1 : Window
    {
        Button button1 = new Button();
        DockPanel dockPanel = new DockPanel();
        public Window1()
        {
            InitializeComponent();
            dockPanel.Background = Brushes.Red;
            dockPanel.LastChildFill = false;
            this.AddChild(dockPanel);
```

```
            button1.Background = Brushes.AliceBlue;
            button1.Height = 50;
            button1.Width = 100;
            button1.Content = content;
            DockPanel.SetDock(button1, Dock.Bottom);
            this.dockPanel.Children.Add(button1);
        }
}
```

Avec WPF :

```
<Window x:Class="WpfApplication1.Window2bis"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window2" Height="300" Width="300">
        <DockPanel Name="dockPanel" LastChildFill="False">
            <Button Name="button1" DockPanel.Dock="Bottom" Width="100"
            Height="50" Background="AliceBlue" Content="Hello"/>
        </DockPanel>
</Window>
```

En WPF, chaque fichier XAML est associé à un fichier en C♯, en effet les balises telles que DockPanel ou Button créent automatiquement ces objets dans le code C♯ on peut donc les modifier à travers lui. Ce fichier C♯ est chargé de la réception des événements sur la fenêtre et les objets qu'elle contient.

3.3 Microsoft Surface

3.3.1 La table tactile

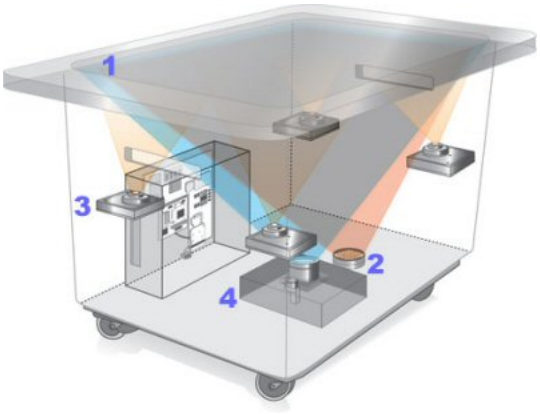


FIGURE 3.2 – Structure de la table tactile

La table tactile de Microsoft sur laquelle tourne le système d'exploitation Microsoft Surface est une simple table recouverte d'une vitre en plexiglas. Elle contient uniquement un vidéoprojecteur permettant de projeter l'interface utilisateur sur la vitre et de quatres caméras infrarouge qui filment celle-ci dans le but de détecter les contacts (Voir image 3.2). Cette table est multi-touch, en effet elle peut reconnaître plus de 52 points de contacts simultanés mais elle peut aussi reconnaître des objets par la forme de leur ombre ou en leur appliquant un *tag* c'est à dire une petite image qui permet à l'objet d'être reconnu. De plus, elle peut communiquer avec les objets qui possédent le bluetooth ou le wifi (d'après le site officiel de Microsoft Surface [5]).

3.3.2 Microsoft Surface

Microsoft Surface (voir image 3.3) est un système d'exploitation basé sur Windows Vista SP1. Il offre une interface adaptée à l'utilisation de la table tactile notamment avec des contrôles utilisateurs spécifiques.

Pour développer sur Surface, il faut installer Microsoft Surface SDK qui s'intègre à Visual Studio, Il faut aussi installer le simulateur qui permet de tester les applications Surface à l'aide de la souris.

FIGURE 3.3 – Le logo de Microsoft Surface

3.4 Autres outils

3.4.1 Microsoft Office OneNote 2007

Microsoft Office OneNote est un logiciel destiné à la prise de notes pouvant être utilisé sur les Tablets PC ou sur les PC normaux. Ce logiciel m'a permis de prendre chaque jour des notes sur l'avancement de mes travaux et de répertorier les documents que j'ai utilisés. Il permet d'organiser clairement et de façon hiérarchique ses notes.

3.4.2 Microsoft Office Visio 2007

Microsoft Visio est un logiciel de diagrammes et de schémas pour Microsoft Windows. J'ai utilisé ce logiciel pour créer les diagrammes UML car je pense qu'il respecte les standard UML et qu'il est pratique à utiliser.

Chapitre 4

Ergonomie

4.1 Interface sous Microsoft Surface

La particularité de Microsoft Surface et qu'il est utilisé sur une table tactile, il faut donc adapter l'interface utilisateur à cet environnement. Pour cela le **SDK** de Microsoft nous fournit plusieurs éléments adaptés, comme des boutons, des boites d'éditions avec claviers virtuels, etc.

De plus, notre jeu s'adresse à de très jeunes enfants, il faut donc une interface très simple et facile à utiliser. Grâce au tactile nous pouvons faire en sorte que l'interaction entre le jeu et l'enfant se fasse de manière très naturelle.

Après réflexion notamment sur la disposition du menu, nous avons trouvé deux interfaces différentes.

4.1.1 Interface n° 1

La première s'inspire d'un jeu de Microsoft Surface « ScatterPuzzle », elle est composée d'un menu en bas où l'on sélectionne la forme que l'on veut faire ainsi que les paramètres. Ensuite, celle-ci s'affiche sur l'écran principal. (voir image 4.1)

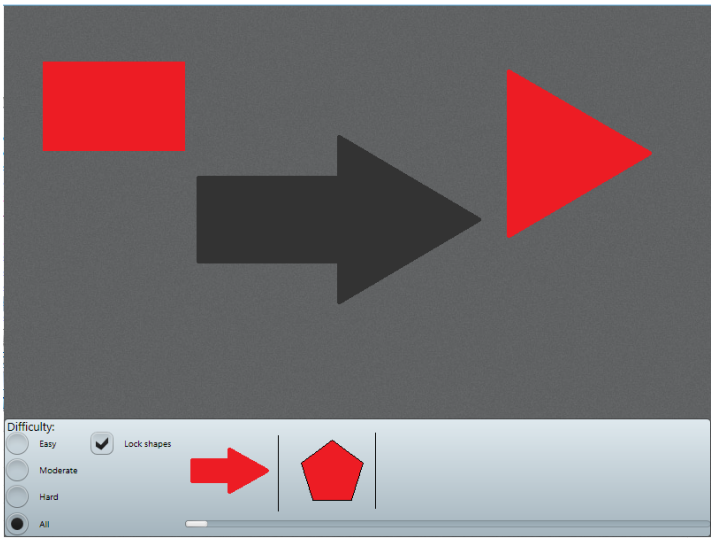


FIGURE 4.1 – Interface n° 1

4.1.2 Interface n° 2

La deuxième est un enchaînement de panneaux, le premier panneau est constitué d'un menu (voir image 4.2(a)) permettant de régler tous les paramètres, le second de toutes les formes que

l'on peut choisir (voir image 4.2(b)) et le troisième de celle que l'on a choisie de reconstituer (voir image 4.2(c)).

4.1.3 Choix

Pourquoi avoir choisi la deuxième interface ?

Après avoir testé le jeu ScatterPuzzle, nous nous sommes aperçus que lorsque nous faisions un puzzle il nous arrivait de toucher le panneau latéral qui permet de changer de puzzle. Ceci change donc le puzzle en cours de partie. On a donc pensé que s'il y a quatre ou cinq enfants qui jouent, le jeu risque d'être vite injouable. Nous avons donc opté pour la deuxième interface qui permet par exemple au professeur de régler les paramètres, puis de laisser ensuite le choix aux enfants de choisir la forme qu'ils veulent.

4.2 Design participatif

Qu'est ce que le design participatif ?

D'après Wikipedia [2], la conception participative est une approche de conception qui vise à impliquer activement tous les utilisateurs dans le processus de conception afin de s'assurer que le produit qui est conçu, répond à leurs besoins et est utilisable. Le terme est utilisé dans une variété de domaines, par exemple la conception de logiciels, le design urbain, architecture, architecture du paysage, la conception du produit, la durabilité, la planification ou encore la médecine comme un moyen de créer des environnements qui sont plus sensibles et adaptés à leurs habitants et usagers culturels, émotionnels, spirituels et pratiques.

Comment avons-nous appliqué ce design participatif ?

À la moitié du stage, des enfants sont venus tester une première version de notre logiciel sur la table Microsoft Surface. Lors de l'utilisation, nous avons pu observer que les enfants n'ont pas besoin qu'on leur explique le jeu, en effet, ils comprennent directement ce qu'ils doivent faire. C'est une très bonne chose qui est rendue possible par le tactile car tout est intuitif, mais aussi grâce à l'ergonomie de notre interface.

Nous avons donc pu en déduire que notre interface est ergonomique.

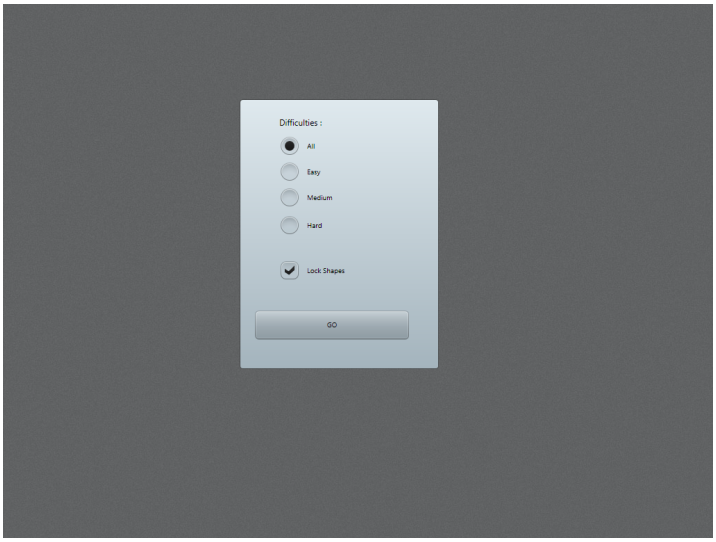
4.3 Autres aspects ergonomiques

Les formes et couleurs

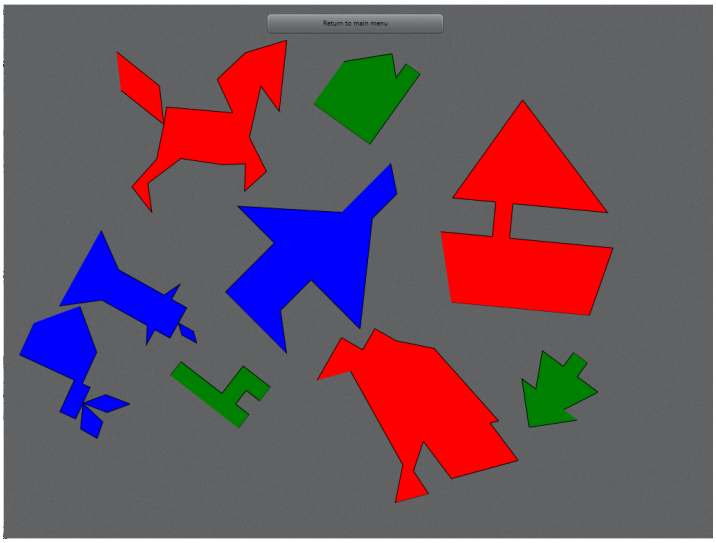
Nous avons mis en place des couleurs par rapport au niveau de difficulté des formes, vert pour facile, bleu pour moyen et enfin rouge pour difficile. Ceci permet d'identifier plus facilement les formes grâce à un repère visuel aisément compréhensible par les enfants. De plus chacune des formes reprèsente un objet ou un animal ce qui rend le jeu plus amusant pour les enfants qui essayent de découvrir ce que représente chaque forme.

Éviter le texte

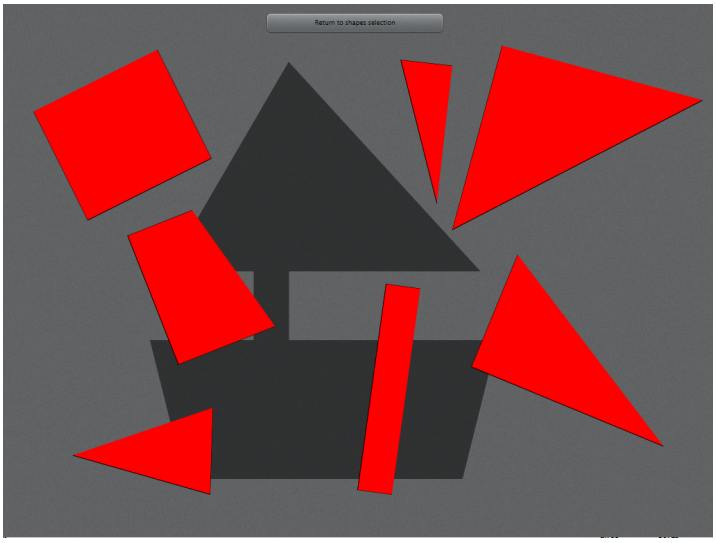
Étant donné que l'application est faite pour les enfants nos tuteurs nous ont conseillé d'intègrer le moins de texte possible car les très jeunes enfants ne savent pas encore lire. De plus cela diminuerait le côté intuitif de l'interface.



(a) Le menu



(b) Choix de la forme



(c) Reconstitution de la forme

FIGURE 4.2 – Interface n° 2

Chapitre 5

Analyse et Programmation

5.1 Vue d'ensemble

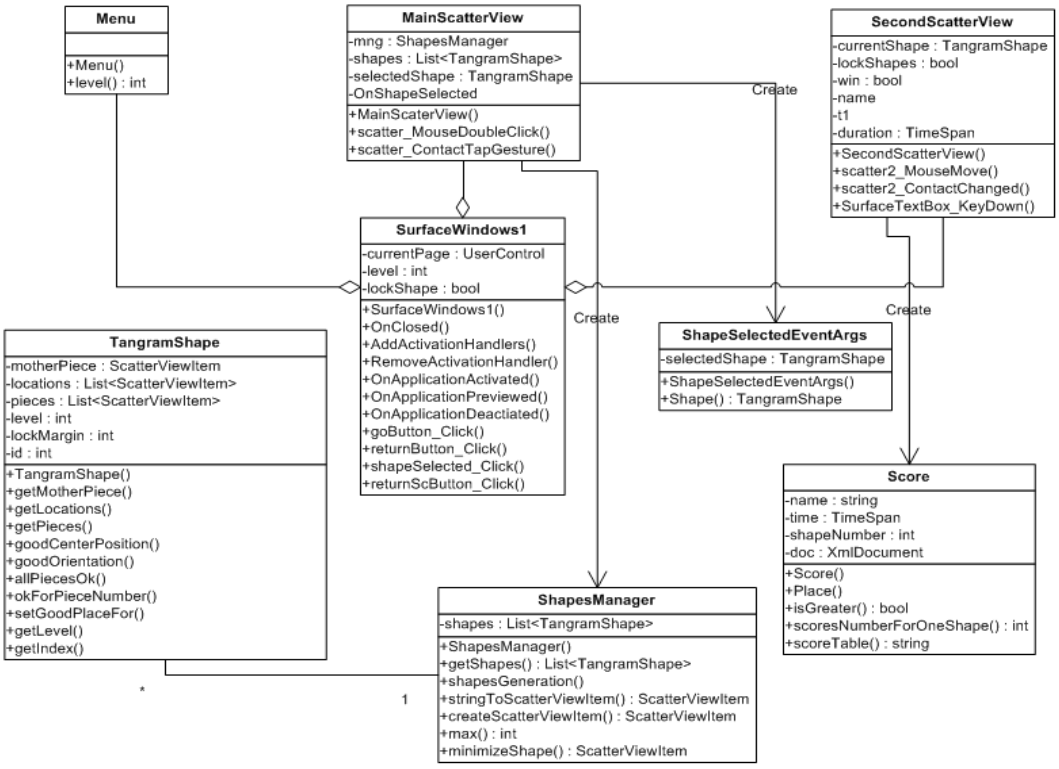


FIGURE 5.1 – Diagramme de classe

Il y a huit classes dans notre programme, cinq pour l'interface et trois pour le moteur. Pour chaque classe de l'interface, il y a un fichier XAML qui y est associé (voir chapitre 5.3). *SurfaceWindows1* représente la fenêtre tandis que *Menu*, *MainScatterView* et *SecondScatterView* représentent les pages. Enfin *ShapeSelectedEventArgs* est un événement (voir chapitre 5.3) qui est déclenché lorsqu'on clique sur une forme dans la page *MainScatterView*.

Du côté du moteur nous trouvons uniquement trois classes, la principale *TangramShape* (voir chapitre 5.7) qui se charge de gérer les formes et leurs pièces, la classe *ShapeManager* (voir chapitre 5.6.3) qui s'occupe de transformer les données sauvegardées en objet *TangramShape* et enfin la classe *Score* (voir chapitre 5.8) qui se charge de sauvegarder les scores des joueurs dans un fichier.

5.2 ScatterView

La scatterview est une des principales interfaces, proposées par le Microsoft Surface SDK. Elle permet de déplacer, effectuer une rotation, agrandir ou rétrécir des objets sur la table tactile. Le meilleur exemple de son utilisation est dans l'application Microsoft Surface Photos qui permet de partager photos et vidéos sur la table comme s'il s'agissait d'une vraie table, on peut les bouger, les tourner, les agrandir, etc.

Après avoir commencé à étudier les technologies que je devais employer, j'ai essayé une première piste pour la réalisation du logiciel en décidant d'utiliser la ScatterView. Grâce à celle-ci les éléments que j'ajouterai pourront déjà être déplacés de façon tactile et de plus elle possède un système de rotation implémenté.

Cependant, il y a un premier problème, par défaut les éléments que l'on place dans une scatterview sont des rectangles comme les images, les vidéos, etc. Il a donc fallu que je trouve un moyen pour qu'elle puisse accepter des objets de différentes formes.

Après quelques recherches sur internet notamment sur le site de Guillaume André [3], j'ai trouvé un moyen de supprimer complètement le style des objets de la ScatterView, de manière à pouvoir appliquer mon propre style c'est-à-dire les formes polygonales. Pour cela il faut redéfinir le style des éléments de la ScatterView (les ScatterViewItem) pour prendre en compte les contours des objets. Il faut rajouter ce code dans la partie ressources de chaque fichier XAML qui possède une ScatterView, vous pourrez le trouver dans l'annexe A.

5.3 Interface avec WPF

WPF permet de voir la programmation autrement, en séparant complètement les aspects graphiques du moteur. C'est une autre façon de programmer, et il m'a fallu un certain temps avant de comprendre son fonctionnement. En effet, dans les premières versions du logiciel j'utilisais le C♯ et XAML avec mon expérience Java, c'est-à-dire en utilisant un minimum le XAML, seulement pour la première page de l'interface, je changeais les objets grâce au C♯. En observant d'autres logiciels en WPF, je me suis aperçu que l'on pouvait faire tout le graphique en XAML, en créant plusieurs pages à la manière d'un site web. En WPF, chaque fichier XAML est associé à un fichier C♯, le fichier XAML s'occupe de la partie graphique et le fichier C♯ de réceptionner les événements qui peuvent être déclenchés dans l'interface.

Dans notre interface nous avons donc trois pages successives, le menu, le choix de la forme et le jeu à proprement parler. Il y a donc un fichier XAML pour chacune de ces pages et un autre dit "principal" qui se charge de faire le lien entre celles-ci (voir image 5.2).

Comment se font les enchaînements entre les différentes pages ?

Les fichiers XAML se chargent uniquement de l'affichage, ce sont les fichiers C♯ qui s'occupent de changer les pages suivant les événements qui se produisent.

C'est la page principale qui possède les éléments qui permettent de passer d'une page à l'autre, c'est-à-dire le bouton "OK" pour le menu, le bouton de retour au menu pour la MainScatterView et le bouton de retour à la MainScatterView pour la SecondScatterView. C'est donc le fichier C♯ qui lui correspond, qui écoute tous les événements relatifs à ces objets, ainsi qu'un autre événement que j'ai créé, qui se déclenche quand l'utilisateur clique sur une forme sur la deuxième page. Ce fichier se charge donc pour chaque événement de charger les pages qui lui correspondent.

Création de l'événement pour la sélection d'une forme

Tout d'abord, il faut créer un événement, pour cela on crée une classe qui hérite de la classe

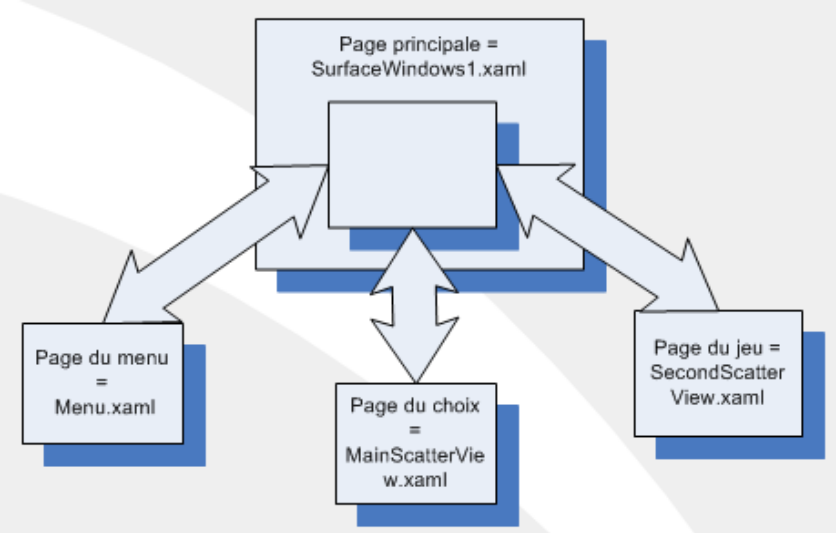


FIGURE 5.2 – Les fichiers XAML

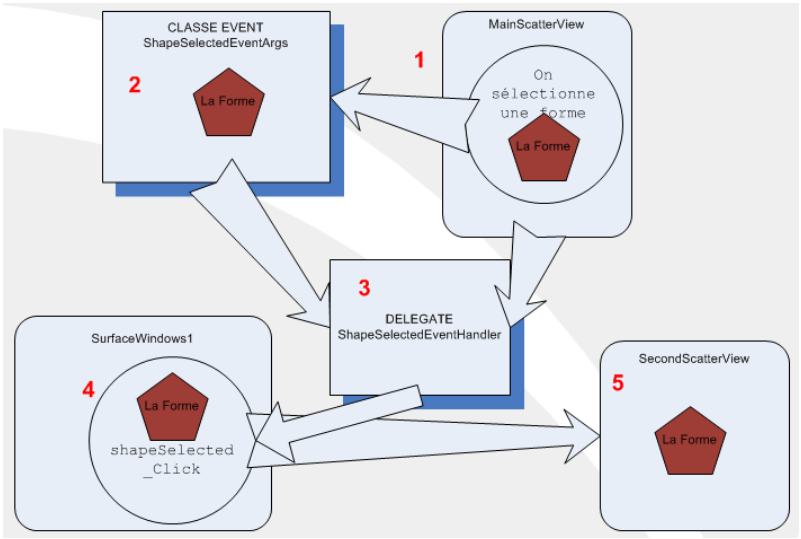


FIGURE 5.3 – La création d'événement en C♯

EventsArgs, on envoie à cet événement la forme qui a été sélectionnée de manière à pouvoir la récupérer dans le fichier *SecondScatterView*.

Pour gérer le déclenchement de l'événement, il faut créer un *delegate*, qui est un objet qui permet d'appeler des fonctions. Il faut donc ensuite lui envoyer la référence de la fonction que l'on veut qu'il appelle. Pour cela on crée donc le delegate qui est un objet de type *ShapeSelectedEventHandler* puis on lui ajoute la fonction que l'on veut ici c'est *shapeSelectedClick* qui se trouve dans *SurfaceWindows1*.

Lorsque que l'on sélectionne une forme (voir image 5.3), un événement est envoyé à la fonction *ContactTapGesture* de la *MainScatterView* cette fonction crée un événement *ShapeSelectedEventArgs* en lui envoyant la forme sélectionnée puis elle envoie l'événement au delegate qui se charge de l'envoyer à toutes les fonctions qui ont été inscrites dans sa liste. Ici il n'y a que la fonction *ShapeSelected* de *SurfaceWindows1*. Il lui envoie donc l'événement puis celle-ci envoie à la *SecondScatterView* la forme. On peut donc dire que ce sont les fichiers C♯ qui se chargent de passer d'une page à l'autre (voir image 5.4).

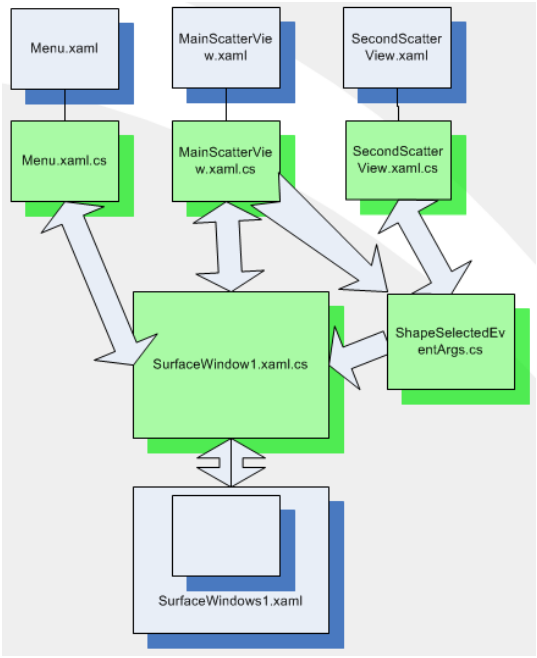


FIGURE 5.4 – Les fichiers XAML et C♯ de l'interface

5.4 Le tactile

Les différences entre le tactile et le non-tactile sont importantes au niveau de l'interface, mais au niveau de la programmation du moteur les différences sont minimes. En effet, elles se situent juste au niveau des événements, les événements souris tels que le double clic (*MouseDoubleClick*) est remplacé par un événement de contact (*ContactTapGesture*) et les mouvements de souris (*MouseMove*) par un autre événement de contact (*ContactChanged*).

5.5 Le mouvement : *ContactChanged*

Dans le jeu à proprement parler c'est-à-dire dans la *SecondScatterView*, à chaque fois que l'on déplace son doigt sur l'écran un événement est déclenché et appel la méthode *Contact-Changed*. C'est elle qui se charge d'identifier ce que déplace le joueur et qu'est ce que ça a comme conséquence dans le jeu.

Il commence par regarder si le jeu n'est pas fini donc si le booléen "win" n'est pas vrai. Ensuite, il regarde dans toutes les pièces s'il y en a une qui est active. Si oui, alors on regarde si elle est la bonne place, si c'est le cas on la verrouille sur celle-ci. Après avoir passer toutes les pièces on regarde si elles sont toutes à la bonne place si oui alors la partie est gagnée, on affiche le logo "winner", le temps et la textbox pour indiquer son nom.

5.6 Gestion des données

5.6.1 Quelles données ?

Pour savoir quelle données il faut conserver il faut savoir comment créer les formes, pour cela nous avons étudié une application qui possède des points communs avec l'application que l'on veut créer, ScatterPuzzle. En effet, c'est une application dans laquelle le joueur doit construire des puzzles. Les pièces de ce puzzle sont dans une ScatterView donc nous avons étudié ce logiciel pour savoir comment les pièces du puzzle étaient créées.

Il en est ressorti que les pièces sont faites grâce à des objets "Path". Ceux-ci peuvent créer toutes sortes de formes quand on leur fournit des points.

Donc les données qu'il va falloir sauvegarder ce sont ces points pour la forme principale ainsi que pour chacune des pièces. De plus, il faut aussi sauvegarder le niveau de difficultés de la forme c'est-à-dire si elle est facile, moyenne ou difficile, enfin il nous faut le centre et l'orientation de chaque pièce pour pouvoir savoir si elles sont à la bonne place (voir chapitre 5.7).

5.6.2 Sauvegarde des données (du .txt au .xml)

Le problème de sauvegarde des données est apparu à partir du moment où nous avons dû ajouter plusieurs formes au jeu. Créer les formes directement avec ces données dans le programme est possible mais d'une part cela rend le code moins clair et d'autre part cela rend plus difficile l'ajout ou la suppression de formes. Il a donc fallu trouver une solution à ce problème, nous avons commencé par une solution que l'on connaissait, le fichier texte, puis nous avons trouvé un autre moyen beaucoup plus efficace le XML. Enfin il nous restait une troisième solution la base de données, mais comme nous n'avions pas un nombre important de données à conserver, cela paressait inutile. Je vais donc exposer ici uniquement les méthodes du fichier texte et du XML.

5.6.2.1 Dans un fichier texte

Nous avons donc dans un premier temps décidé de sauvegarder les données dans un fichier texte simple. Il nous a donc fallu créer un fichier texte avec certains repères comme un "c" pour le centre, un "o" pour l'orientation et un nombre d'espaces précis (voir image 5.5). Cela nous a donc permis de pouvoir nous repérer dans le fichier texte au niveau du traitement des données (voir chapitre 5.6.3).

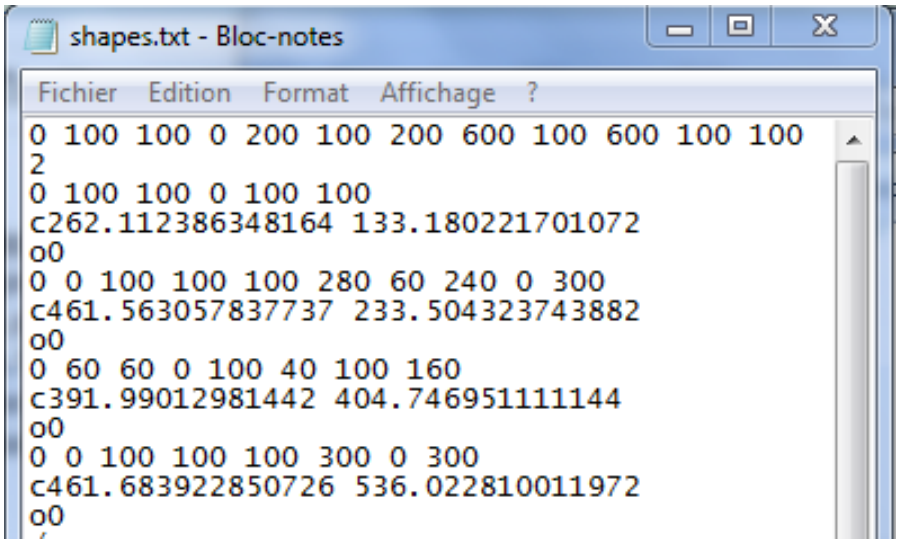


FIGURE 5.5 – Le fichier .txt

5.6.2.2 Dans un fichier XML

Quelques temps plus tard nous nous sommes aperçus qu'il y avait un autre moyen beaucoup plus pratique de sauvegarder les données de manière "standardiser" le XML. En effet, dans un fichier XML les données sont placées hiérarchiquement dans des balises et des attributs (voir image 5.6), de plus il existe un certain nombre de classes et méthodes permettant de traiter ses

données (voir chapitre 5.6.3). La notation en code XML permet donc une meilleure lisibilité des données et une réutilisation de celles-ci dans différentes applications.

shapes.xml

```
<TangramShape Level="1" MotherPiece="0 0 50 0 50 150 150 150 150
              250 100 250 100 200 50 200 50 250 0 250">
  <Piece Orientation="0" PositionX="512,210119664609"
         PositionY="384,034024432058" Path="0 0 50 0 50 250 0 250"/>
  <Piece Orientation="0" PositionX="463,089143656964"
         PositionY="433,784835037688" Path="0 0 50 0 50 50 0 50"/>
  <Piece Orientation="0" PositionX="538,116100139913"
         PositionY="458,741127082697" Path="0 0 50 0 50 100 0 100"/>
</TangramShape>
```

FIGURE 5.6 – Le fichier .xml

5.6.3 Traitement des données(*ShapesManager*)

Une autre partie du logiciel consiste à effectuer le traitement des données, c'est-à-dire convertir les données concernant les formes en objets.

Toutes ces données sont sauvegardées dans un fichier XML, la classe *ShapesManager* est donc chargée, à chaque lancement du logiciel, d'extraire ces données du fichier XML pour pouvoir créer les objets de type *TangramShape* leur correspondant. Pour cela j'utilise le package "System.xml" du framework .Net qui possède donc des classes et méthodes permettant de traiter les données d'un fichier XML.

Nous allons d'abord voir comment nous procédions avec l'ancienne version du fichier texte puis nous verrons la procédure en XML.

5.6.3.1 Comment récupère-t-on les données d'un fichier texte ?

Tout d'abord, le fichier texte doit être écrit de façon très stricte, un espace en plus et toute l'application plante. En effet, quand on va devoir parcourir le fichier il va falloir se repérer par rapport aux signes que l'on a mis en place de façon "artisanale". Nous devons donc parcourir le fichier ligne par ligne en prenant en compte chacun de ces signes. Pour cela on utilise un objet de type "StreamReader" qui contient le fichier texte. Avec la méthode "ReadLine" on obtient les lignes sous forme de "String" les unes après les autres. Étant donné que les "String" sont des tableaux de caractères, on vérifie si le premier caractère n'est pas "c" ou "o", ce sont donc des coordonnées que l'on transforme en "ScatterViewItem" (voir chapitre 5.6.3.3).

Exemple :

```
/**
* result: return a list with all the pieces in the selected file.
**/
private List<ScatterViewItem> piecesGeneration(StreamReader readPieces)
{
 List<ScatterViewItem> pieces = new List<ScatterViewItem>();
 String line = readPieces.ReadLine();
 while (line != null && line != "" && line != "/")
 {
        if (line[0] != 'c' && line[0] != 'o')
        {
                ScatterViewItem piece = this.stringToScatterViewItem(line);
        pieces.Add(piece);
```

```
        }
 line = readPieces.ReadLine();
 }
 return pieces;
}
```

Vous pourrez trouver en annexes le code de la classe ShapesManagers avec les fichiers texte.

5.6.3.2 Comment récupère-t-on les données d'un fichier XML ?

Tout d'abord, on crée un objet de type "XmlDocument" puis l'on charge notre fichier XML dans cet objet grâce à la méthode "Load". Ensuite on peut parcourir notre fichier comme une liste de nœud ("node" en anglais) car en XML chaque couple de balise est un nœud (exemple : <nœud></nœud>), on peut donc parcourir chacun de ces nœuds grâce à une simple boucle "foreach" comme celle-ci :

```
foreach (XmlNode n in doc.DocumentElement.ChildNodes)
 {

 }
```

Puis il suffit d'accéder aux attributs du nœud en cours grâce à la commande "n.Attributes["Nom de l'attribut"].Value" On peut donc dire qu'il est beaucoup plus simple d'utiliser XML pour traiter les données, car on accède directement à l'information que l'on veut.

Vous pourrez trouver le code de ShapesManager avec les fichiers XML en annexes de manière à pouvoir comparer avec le fichier texte.

5.6.3.3 La conversion des données en TangramShape

Il faut maintenant convertir ces données qui sont des "String" dans le fichier de sauvegarde en objets de type *TangramShape*.

De la chaîne de caractères à la liste de coordonnées

Pour créer la liste des coordonnées on parcourt la "String", on enregistre tous les caractères jusqu'à ce qu'on tombe sur un espace. On convertit alors ce groupe de caractère en nombre puis on l'enregistre dans la liste. On continue comme ceci jusqu'à la fin de la chaîne.

De la liste des coordonnées à l'objet "Path"

On possède donc une liste de coordonnées que l'on a extraite du fichier XML dans l'attribut "path".

On crée donc un objet "PathFigure" et on définit le point de départ de la forme, ici ce sont les deux premières coordonnées de la liste.

Ensuite, étant donné que l'on veut tracer un polygone, on crée un objet qui est une collection de segments de type "PathSegmentCollection". Puis on parcourt toute la liste et pour chaque couple de coordonnées, on crée un objet "LineSegment" auquel on ajoute un point avec ces coordonnées. Enfin on ajoute ce "LineSegment" à la collection de segment. De cette manière le "PathFigure" va savoir que de son point de démarrage il doit aller au point suivant en traçant un segment, puis au point suivant, etc.

Une fois ceci fait, on ajoute les segments à la propriété "Segments" du "PathFigure" puis on inclut celui-ci dans un "PathFigureCollection" pour le cas où on voudrait avoir plusieurs figures. Enfin la collection de "PathFigure" est ajouté dans un "PathGeometry".

On peut donc créer le "Path" en lui ajoutant à la propriété "Data" ce "PathGeometry" (voir image 5.7). On peut aussi modifier certains paramètres comme la couleur de fond, l'épaisseur des traits, etc.

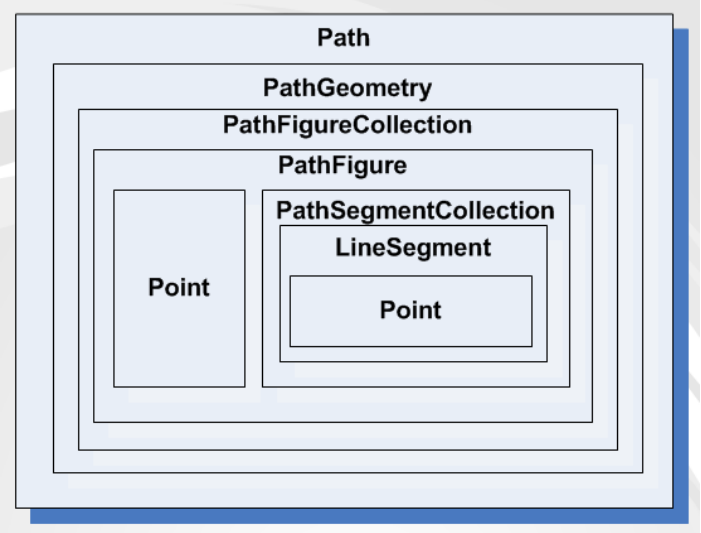


FIGURE 5.7 – Création du Path

Du "Path" au "ScatterViewItem"

Pour finir, il ne reste plus qu'à créer un objet "ScatterViewItem", auquel on ajoute le "Path" dans la propriété "Content" puis on définie la taille du "ScatterViewItem" pour que la forme soit représentée correctement. Nous avons maintenant plus qu'à faire ceci pour la pièce mère et les petites pièces pour créer les *TangramShapes*.

5.6.3.4 La miniaturisation des formes pour le menu (*minimizeShape*)

Pour afficher les formes dans le menu on doit réduire leurs tailles. Pour cela on utilise la méthode *minimizeShape*, celle-ci s'occupe quand on lui envoie un "ScatterViewItem" d'extraire le "Path" puis d'en recréer un nouveau en prenant tous les points de celui-ci et en les divisant par deux. Puis elle reconstruit le "ScatterViewItem".

5.7 Moteur(*TangramShape*)

Une fois les données traitées on obtient donc une liste de TangramShape qui représente donc chacune des formes du jeu. Un TangramShape est donc constitué de la forme que l'on appellera "mère" c'est-à-dire la forme entière, de toutes les pièces qui composent la forme, ainsi que des emplacements dans lesquels doivent se bloquer les pièces. Bien sûr chacun de ces objets est un élément de la ScatterView.

La classe TangramShape qui possède ces éléments doit donc gérer toutes les interactions de ces éléments dans le déroulement du jeu. Elle doit donc être capable de dire si une pièce est à la bonne place (*okForPieceNumber()*), de bloquer cette pièce si c'est vrai (*setGoodPlaceFor*) et de savoir si toutes les pièces sont à leur place (*allPiecesOk()*) c'est-à-dire si la partie est gagnée.

Comment sait-on si une pièce est à sa place ?

Pour cela nous avons deux choses à vérifier, d'une part si le centre de la pièce est égal à celui de son emplacement (*goodCenterPosition()*), d'autre part est-ce que l'orientation de la pièce correspond à celui de son emplacement (*goodOrientation()*).

L'utilité que les emplacements des pièces soient aussi des éléments de la ScatterView réside dans le fait que l'on peut ainsi comparer simplement le centre et l'orientation des deux pièces.

La marge d'erreur (*LockMargin*)

Que ce soit pour le centre ou l'orientation il y a une marge d'erreur, c'est-à-dire que, même si la pièce n'est pas tout à fait à la bonne place, elle est tout de même bloquée automatiquement à la bonne place. On vérifie donc si le X et le Y du centre de la pièce sont à la même position que le X et le Y de l'emplacement avec plus ou moins une marge d'erreur. Puis on procède de même avec l'orientation de la pièce, il faut qu'elle ait la même orientation que son emplacement avec plus ou moins une marge d'erreur.

Bloquer les pièces

Pour bloquer les pièces, il suffit de mettre à faux certains paramètres des ScatterViewItems, les propriétés "canMove" pour l'empêcher de bouger et "canRotate" pour empêcher la rotation. Enfin on remplace le centre et l'orientation par ceux de l'emplacement qui lui correspond.

5.8 L'enregistrement des scores

Pour enregistrer le temps que met le joueur à réaliser la pièce, on enregistre l'heure quand le joueur lance la forme pour cela on utilise un objet de type *DateTime* sa fonction "now" nous donne l'heure. Ensuite on enregistre l'heure quand le joueur finit le jeu c'est-à-dire au moment où toutes les pièces sont à leur place. Ensuite on utilise la méthode "substract" du *DateTime* pour soustraire à la deuxième date la première comme cela nous avons le temps que met le joueur pour effectuer le jeu. On affiche donc ce score, mais on propose aussi au joueur d'indiquer son nom dans une "SurfaceTextBox", celle-ci est une TextBox spécialement conçue pour surface car quand on la touche elle propose un clavier virtuel pour taper son nom. C'est quand le joueur tape sur "enter" que l'on crée un nouvel objet score avec son nom, son temps et la figure qu'il a finie.

Dans cette classe on ouvre le fichier score.xml pour y enregistrer les données de la même manière que vue dans le chapitre 5.6, cependant là on n'extrait pas les données, on en ajoute pour cela on crée un nouveau noeud, en copiant le premier grâce à la méthode "clone" de ce dernier, puis on remplit les attributs avec les données que l'on nous a envoyées, on utilise la méthode "place" qui va placer notre score dans le fichier avant le score qui lui est inférieur pour la même forme enfin on sauvegarde le fichier score.xml.

Il nous reste plus qu'à afficher la liste des dix meilleurs scores grâce à la méthode "score-table".

5.9 Detection de collision

Malgré le fait que je n'ai pas eu le temps d'implémenter un détecteur de collision, j'ai trouvé plusieurs pistes de recherche. Tout d'abord, j'ai essayé d'en faire un moi-même, mais je me suis vite aperçu que cela dépassait mes compétences et risquait de me prendre longtemps. Ensuite j'ai essayé d'utiliser un moteur de jeu nommé XNA et enfin un tuteur m'a indiqué un blog dans lequel quelqu'un mettait à disposition un générateur physique pour la *ScatterView* qu'il avait développé. Je vais donc développer ci-dessous ces pistes de recherche.

5.9.1 Faire un détecteur de collision

J'ai donc commencé à explorer les différentes techniques de détection de collision, notamment une technique que j'ai trouvée sur le site de CodeProject [1]. Celle-ci consiste à considérer

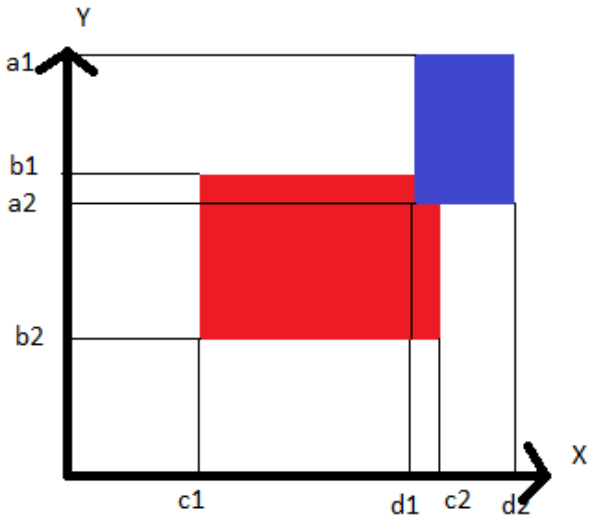


FIGURE 5.8 – Méthode de détection de collision

les formes dans un repère 2D, on projette ensuite la forme sur l'axe X, ensuite on fait pareil avec l'autre forme, puis on compare les projections si elles se chevauchent c'est que les deux formes sont en collision. Pour finir on fait pareil avec l'axe des Y (voir image 5.8).

Cette technique marche bien, mais uniquement avec les formes pas trop complexes, comme les carrés ou rectangles, si l'on prend par exemple des triangles cela devient beaucoup plus complexes les projections peuvent se chevaucher bien que les triangles ne se touchent pas. J'ai donc ensuite décidé sur le conseil de mes tuteurs de rechercher des pistes du côté de la plateforme de jeu XNA.

5.9.2 XNA

XNA est une plateforme de jeu développée par Microsoft pour sa XBOX360 cependant il est aussi possible de l'utiliser pour développer des jeux sur surface. Cette plateforme vise à simplifier la création de jeu. Après plusieurs tutoriels, je me suis rendu compte que ce n'était pas adapté à un jeu aussi simple que le Tangram et que cela me prendrait trop de temps d'apprendre comment cela fonctionnait même s'il devait y avoir un moteur physique. Une troisième piste m'a été envoyée par mes tuteurs, Surface Physics.

5.9.3 Surface Physics

Physics est une librairie créée par le Dr Dave [4], c'est une nouvelle version de la ScatterView, mais avec des propriétés physiques. Après avoir essayé d'étudier le fonctionnement de sa librairie, le problème a été de savoir comment mettre des formes dans la PhysicView de manière à ce qu'elle possède les mêmes propriétés physiques, après quelques recherches infructueuses j'ai décidé de lui poser la question sur son blog sans réponse de sa part et sans aucun autre moyen de le contacter, j'ai dû abandonner.

Chapitre 6

Discussion

6.1 Résultats obtenus et résultats attendus

Quand on compare l'application finale aux critères du cahier des charges (voir chapitre 2.1.2), on peut dire que la majoritée des points ont été implémentés. Le joueur peut choisir une forme parmi plusieurs formes quand il en choisit une, le puzzle apparaît, il peut déplacer les pièces, les faire tourner, on peut choisir en option si les pièces doivent se bloquer ou pas sur leur emplacement, on peut trier les formes selon leurs difficultés et enfin le jeu enregistre le temps que met le joueur pour réaliser le jeu. Le seul point que l'on n'a pas pu implémenter c'est le système physique malgré quelques pistes de recherches (voir chapitre 5.9) je pense que nous manquions de temps et de compétences pour finir cette partie du projet.

6.2 Optimisations possibles du projet

Outre, bien sûr la mise en place d'un moteur physique, je pense qu'une des futurs améliorations de ce projet est la mise en place d'un éditeur de forme. C'est-à-dire qu'un élève pourrait créer sa propre forme et la découper puis pourrait mettre au défit ses amis de la reconstituer. Cette création devrait être gérée de façon tactile et intuitive. Un éditeur permettrait une vraie durée de vie au jeu, car des formes pourraient être ajoutées en permanence. Dans notre version du jeu il y a uniquement un programmeur qui peut rajouter des formes ce qui limite forcément les perspectives de jeu.

Conclusion

Bilan personnel

Ce projet m'a apporté beaucoup de choses, tout d'abord de par le contexte dans lequel je l'ai réalisé, en effet je suis parti à Preston où j'ai dû d'une part m'adapter à la langue et à la culture, mais aussi aux conditions de travail. Comme je n'étais pas en entreprise, il n'y avait pas d'horaire, il fallait que je travaille moi-même que je gère mon propre temps. On peut dire que l'environnement de travail s'apparentait à celui de la recherche.

De plus, j'ai pu me rendre compte à travers ce stage qu'un informaticien est toujours en formation, en effet la plus grosse partie de mon stage a été d'apprendre de nouveaux langages et technologies et de me les approprier.

Ce stage m'a donc apporté énormément de choses que ce soit en qualité d'adaptation, de travail, d'autonomie et d'ouverture d'esprit. Je peux dire que c'est une des meilleures expériences de ma vie.

Bibliographie

[1] CodeProject - Detection de collision. http://www.codeproject.com/KB/GDI-plus/PolygonCollision.aspx.

[2] Design participatif. http://en.wikipedia.org/wiki/Participatory_design.

[3] Guillaume André - Surface : Personnaliser le contrôle ScatterView. http://www.guillaumeandre.com/surface-scatterview-developpement-custom/.

[4] Le blog du Dr Dave - Surface Physics. http://drdave.co.uk/blog/post/Surface-Physics-Download.aspx.

[5] Microsoft Surface - site officiel. http://www.microsoft.com/surface/.

[6] MSDN - Forum. http://social.msdn.microsoft.com/Search/en-US/?Refinement=112&query=surface.

[7] Wikipedia - C♯. http://fr.wikipedia.org/wiki/C_sharp.

[8] Wikipedia - Microsoft .Net. http://fr.wikipedia.org/wiki/Microsoft_.NET.

[9] Wikipedia - Tangram. http://fr.wikipedia.org/wiki/Tangram.

[10] Wikipedia - WPF. http://fr.wikipedia.org/wiki/Windows_Presentation_Foundation.

[11] Wikipedia - XAML. http://fr.wikipedia.org/wiki/XAML.

Annexe A

Le code pour changer le style de la ScatterView

```
<!-- Style for ScatterViewItems -->
   <Style TargetType="{x:Type␣s:ScatterViewItem}">
       <Setter Property="Template">
           <Setter.Value>
               <ControlTemplate TargetType="{
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣x:Type␣s:ScatterViewItem}">
                   <ContentPresenter HorizontalAlignment="{
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣TemplateBinding␣HorizontalContentAlignment}"
                   VerticalAlignment="{
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣TemplateBinding␣VerticalContentAlignment}"
                   SnapsToDevicePixels="{
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣TemplateBinding␣SnapsToDevicePixels}" />
               </ControlTemplate>
           </Setter.Value>
       </Setter>
   </Style>
```

Annexe B

La classe ShapesManager avec un fichier texte

```
class ShapesManager
 {
     List<TangramShape> shapes;
     TextBox r;

     public ShapesManager(TextBox r)
     {
         this.r = r;
         this.r.Text = "";
         this.shapes = new List<TangramShape>();
         this.shapesGeneration();
     }

     public List<TangramShape> getShapes()
     {
         return this.shapes;
     }

     /**
      * action: create the list of all the tangramShape
      from a textfile.
      **/
     public void shapesGeneration()
     {

         this.shapes = new List<TangramShape>();
         StreamReader readShapesPiece = null;
         StreamReader readShapesLocation = null;
         string line;

         if (File.Exists("shapes.txt"))
         {
             readShapesPiece = new StreamReader(
             "shapes.txt");
             readShapesLocation = new StreamReader(
             "shapes.txt");
             line = readShapesPiece.ReadLine();
             readShapesLocation.ReadLine();
```

```
            while (line != null)
            {
                //the motherpiece
                ScatterViewItem motherPiece =
                this.stringToScatterViewItem(line);
                line = readShapesPiece.ReadLine();
                readShapesLocation.ReadLine();

                //level
                String t = "";
                t += line[0].ToString();
                int level = Convert.ToInt32(t);

                //pieces
                List<ScatterViewItem> pieces =
                this.piecesGeneration(readShapesPiece);

                //locations
                List<ScatterViewItem> locations =
                this.locationGeneration(readShapesLocation);

                foreach (ScatterViewItem loc in locations)
                {
                    System.Windows.Shapes.Path p = (
                    System.Windows.Shapes.Path)loc.Content;
                    p.Fill = new SolidColorBrush(
                    Colors.Black);
                    loc.Opacity = 0.5;
                    loc.CanMove = loc.CanRotate =
                    loc.CanScale = false;
                }

                //create the TangramShape
                this.shapes.Add(new TangramShape(
                motherPiece, locations, pieces, level));

                line = readShapesPiece.ReadLine();
                readShapesLocation.ReadLine();
            }
        }

    }

    /**
     * result: return a list with all the pieces
     in the selected file.
     **/
    private List<ScatterViewItem> piecesGeneration(
    StreamReader readPieces)
    {
        List<ScatterViewItem> pieces =
        new List<ScatterViewItem>();
        String line = readPieces.ReadLine();
        while (line != null && line != "" && line != "/")
```

```
        {
            if (line[0] != 'c' && line[0] != 'o')
            {
                ScatterViewItem piece =
                this.stringToScatterViewItem(line);
                pieces.Add(piece);
            }
            line = readPieces.ReadLine();
        }
        return pieces;
    }

    private List<ScatterViewItem> locationGeneration(
    StreamReader readPieces)
    {
        List<ScatterViewItem> locations =
        new List<ScatterViewItem>();
        String line = readPieces.ReadLine();
        ScatterViewItem piece = null;

        while (line != null && line != "" && line != "/")
        {
            if (line[0] != 'c' && line[0] != 'o')
            {
                piece = this.stringToScatterViewItem(line);
                locations.Add(piece);
            }
            else if (line[0] == 'c')
            {
                if (piece != null)
                {
                    piece.Center = this.stringToPoint(line);
                }
            }
            else if (line[0] == 'o')
            {
                if (piece != null)
                {
                    Char[] n = line.ToCharArray();
                    n[0] = '0';
                    String s = "";
                    foreach (char c in n)
                    {
                        s += c;
                    }
                    piece.Orientation = Convert.ToDouble(s);
                }
            }
            line = readPieces.ReadLine();
        }
        return locations;
    }

    public ScatterViewItem stringToScatterViewItem(
```

```
        String line)
        {
            List<int> tab = new List<int>();
            String num = "";

            foreach (char c in line)
            {
                if (c != ' ')
                {
                    num += c;
                }
                else
                {
                    tab.Add(Convert.ToInt32(num));
                    num = "";
                }
            }
            return this.createScatterViewShape(tab);
        }

        public Point stringToPoint(String line)
        {
            List<Double> tab = new List<Double>();
            String num = "";

            foreach (char c in line)
            {
                if (c != ' ' && c != 'c')
                {
                    num += c;
                }
                else if (c == ' ')
                {
                    tab.Add(Convert.ToDouble(num));
                    num = "";
                }
            }

            return new Point(tab[0], tab[1]);
        }

        public ScatterViewItem createScatterViewShape(
        List<int> coord)
        {
            ScatterViewItem myShape = new ScatterViewItem();
            //create the pathGeometry
            if (coord != null)
            {
                PathFigure myPathFigure = new PathFigure();
                myPathFigure.StartPoint = new Point(
                coord[0], coord[1]);

                PathSegmentCollection myPathSegmentCollection =
                new PathSegmentCollection();
```

```
            for (int i = 2; i < coord.Count(); i = i + 2)
            {
                LineSegment myLineSegment = new LineSegment();
                myLineSegment.Point = new Point(
                coord[i], coord[i + 1]);
                myPathSegmentCollection.Add(myLineSegment);
            }

            myPathFigure.Segments = myPathSegmentCollection;
            PathFigureCollection myPathFigureCollection =
            new PathFigureCollection();
            myPathFigureCollection.Add(myPathFigure);
            PathGeometry myPathGeometry = new PathGeometry();
            myPathGeometry.Figures = myPathFigureCollection;

            //create the path
            System.Windows.Shapes.Path myPath =
            new System.Windows.Shapes.Path();
            myPath.Stroke = Brushes.Black;
            myPath.StrokeThickness = 1;
            myPath.Data = myPathGeometry;
            myPath.Fill = new SolidColorBrush(Colors.Blue);

            //create the scatterViewShape
            myShape.Content = myPath;
            myShape.Width = this.max(coord);
            myShape.Height = this.max(coord);
        }

        return myShape;
    }

    public int max(List<int> tab)
    {
        int max = 0;
        for (int i = 0; i < tab.Count(); i++)
        {
            if (max < tab[i])
            {
                max = tab[i];
            }
        }

        return max;
    }

    public bool isSite(int p, int p2)
    {
        return this.shapes[p].okForPieceNumber(p2);
    }

    public ScatterViewItem minimizeShape(ScatterViewItem myShape)
    {
        //extract the path
```

```
            System.Windows.Shapes.Path myPath = (
            System.Windows.Shapes.Path)myShape.Content;
            //extract the PathGeometry
            PathGeometry myPathGeometry =
            myPath.Data.GetFlattenedPathGeometry();
            //extract the PathFigure
            PathFigure myPathFigure =
            myPathGeometry.Figures[0];

            //create a new PathFigure
            PathFigure myNewPathFigure =
            new PathFigure();
            Point sp = myPathFigure.StartPoint;
            myNewPathFigure.StartPoint =
            new Point(sp.X / 2, sp.Y / 2);

            PathSegmentCollection myNewPathSegmentCollection =
            new PathSegmentCollection();
            foreach (PolyLineSegment l in myPathFigure.Segments)
            {
                PolyLineSegment myPolyLineSegment =
                new PolyLineSegment();
                foreach (Point p in l.Points)
                {
                    Point np = new Point(p.X/2, p.Y/2);
                    myPolyLineSegment.Points.Add(np);
                }
                myNewPathSegmentCollection.Add(myPolyLineSegment);
            }

            myNewPathFigure.Segments = myNewPathSegmentCollection;

            PathFigureCollection myNewPathFigureCollection =
            new PathFigureCollection();
            myNewPathFigureCollection.Add(myNewPathFigure);
            PathGeometry myNewPathGeometry = new PathGeometry();
            myNewPathGeometry.Figures = myNewPathFigureCollection;

            //create the new path
            System.Windows.Shapes.Path myNewPath =
            new System.Windows.Shapes.Path();
            myNewPath.Stroke = Brushes.Black;
            myNewPath.StrokeThickness = 1;
            myNewPath.Data = myNewPathGeometry;
            myNewPath.Fill = myPath.Fill;

            //create the new scatterViewItem
            ScatterViewItem myNewShape = new ScatterViewItem();
            myNewShape.Content = myNewPath;
            myNewShape.Width = myShape.Width/2;
            myNewShape.Height = myShape.Height/2;

            return myNewShape;
        }
```

```
    }
```

Annexe C

La classe ShapesManager avec un fichier XML

```
class ShapesManager
    {
        List<TangramShape> shapes;

        public ShapesManager()
        {
            this.shapes = new List<TangramShape>();
            this.shapesGeneration();
        }

        public List<TangramShape> getShapes()
        {
            return this.shapes;
        }

        /**
         * action: create the list of all the tangramShape from a
         xml file.
         **/
        public void shapesGeneration()
        {
            XmlDocument doc = new XmlDocument();
            doc.Load(@"../../shapes_En.xml");
            int i = 0;
            foreach (XmlNode n in doc.DocumentElement.ChildNodes)
            {
                i++;
                ScatterViewItem motherPiece =
                this.stringToScatterViewItem(
                n.Attributes["MotherPiece"].Value);
                int level = Convert.ToInt32(
                n.Attributes["Level"].Value);
                List<ScatterViewItem> pieces =
                new List<ScatterViewItem>();
                List<ScatterViewItem> locations =
                new List<ScatterViewItem>();

                foreach (XmlNode e in n.ChildNodes)
```

```
            {
                ScatterViewItem p = this.stringToScatterViewItem(
                e.Attributes["Path"].Value);
                pieces.Add(p);

                ScatterViewItem l = this.stringToScatterViewItem(
                e.Attributes["Path"].Value);
                double x = Convert.ToDouble(
                e.Attributes["PositionX"].Value);
                double y = Convert.ToDouble(
                e.Attributes["PositionY"].Value);
                l.Center = new Point(x, y);
                l.Orientation = Convert.ToDouble(
                e.Attributes["Orientation"].Value);
                locations.Add(l);
            }

            this.shapes.Add(new TangramShape(
            i, motherPiece, locations, pieces, level));
        }
    }

    public ScatterViewItem stringToScatterViewItem(
    String line)
    {
        List<int> tab = new List<int>();
        String num = "";

        foreach (char c in line)
        {
            if (c != ' ')
            {
                num += c;
            }
            else
            {
                tab.Add(Convert.ToInt32(num));
                num = "";
            }
        }
        tab.Add(Convert.ToInt32(num));
        return this.createScatterViewItem(tab);
    }

    public ScatterViewItem createScatterViewItem(
    List<int> coord)
    {
        ScatterViewItem myShape = new ScatterViewItem();
        //create the pathGeometry
        if (coord != null)
        {
            PathFigure myPathFigure = new PathFigure();
            myPathFigure.StartPoint = new Point(
            coord[0], coord[1]);
```

```
            PathSegmentCollection myPathSegmentCollection =
            new PathSegmentCollection();
            for (int i = 2; i < coord.Count(); i = i + 2)
            {
                LineSegment myLineSegment = new LineSegment();
                myLineSegment.Point = new Point(
                coord[i], coord[i + 1]);
                myPathSegmentCollection.Add(myLineSegment);
            }

            myPathFigure.Segments = myPathSegmentCollection;
            PathFigureCollection myPathFigureCollection =
            new PathFigureCollection();
            myPathFigureCollection.Add(myPathFigure);
            PathGeometry myPathGeometry = new PathGeometry();
            myPathGeometry.Figures = myPathFigureCollection;

            //create the path
            System.Windows.Shapes.Path myPath =
            new System.Windows.Shapes.Path();
            myPath.Stroke = Brushes.Black;
            myPath.StrokeThickness = 1;
            myPath.Data = myPathGeometry;
            myPath.Fill = new SolidColorBrush(Colors.Blue);

            //create the scatterViewItem
            myShape.Content = myPath;
            myShape.Width = this.max(coord);
            myShape.Height = this.max(coord);
        }

        return myShape;
    }

    public int max(List<int> tab)
    {
        int max = 0;
        for (int i = 0; i < tab.Count(); i++)
        {
            if (max < tab[i])
            {
                max = tab[i];
            }
        }

        return max;
    }

    public ScatterViewItem minimizeShape(ScatterViewItem myShape)
    {
        //extract the path
        System.Windows.Shapes.Path myPath = (
        System.Windows.Shapes.Path)myShape.Content;
```

```
            //extract the PathGeometry
            PathGeometry myPathGeometry =
            myPath.Data.GetFlattenedPathGeometry();
            //extract the PathFigure
            PathFigure myPathFigure =
            myPathGeometry.Figures[0];

            //create a new PathFigure
            PathFigure myNewPathFigure =
            new PathFigure();
            Point sp = myPathFigure.StartPoint;
            myNewPathFigure.StartPoint =
            new Point(sp.X / 2, sp.Y / 2);

            PathSegmentCollection myNewPathSegmentCollection =
            new PathSegmentCollection();
            foreach (PolyLineSegment l in myPathFigure.Segments)
            {
                PolyLineSegment myPolyLineSegment =
                new PolyLineSegment();
                foreach (Point p in l.Points)
                {
                    Point np = new Point(p.X/2, p.Y/2);
                    myPolyLineSegment.Points.Add(np);
                }
                myNewPathSegmentCollection.Add(myPolyLineSegment);
            }

            myNewPathFigure.Segments = myNewPathSegmentCollection;

            PathFigureCollection myNewPathFigureCollection =
            new PathFigureCollection();
            myNewPathFigureCollection.Add(myNewPathFigure);
            PathGeometry myNewPathGeometry = new PathGeometry();
            myNewPathGeometry.Figures = myNewPathFigureCollection;

            //create the new path
            System.Windows.Shapes.Path myNewPath =
             new System.Windows.Shapes.Path();
            myNewPath.Stroke = Brushes.Black;
            myNewPath.StrokeThickness = 1;
            myNewPath.Data = myNewPathGeometry;
            myNewPath.Fill = myPath.Fill;

            //create the new scatterViewItem
            ScatterViewItem myNewShape = new ScatterViewItem();
            myNewShape.Content = myNewPath;
            myNewShape.Width = myShape.Width/2;
            myNewShape.Height = myShape.Height/2;

            return myNewShape;
        }
    }
```

Résumé

Imaginez vous dans votre salon, vous êtes en famille et vos enfants veulent jouer à un jeu de société. Au milieu de la pièce, entre votre télé et votre canapé se trouve une table basse, elle semble tout à fait normale...Cependant vous la touchez avec votre index et là une des dernières technologies tactiles démarre : Microsoft Surface. Celle-ci vous offre une multitude d'applications, vous pouvez jouer aux échecs, regarder vos photos, etc. Toute votre famille se rassemble autour de cette technologie. Grâce au tactile l'informatique n'est plus individuel, mais se partage.

Ce projet consiste à créer un jeu pour cette table tactile, c'est un jeu de puzzle nommé Tangram. Dans ce rapport vous verrez entre autres les technologies qui m'ont permis de créer ce jeu notamment le C♯ et les problèmes ergonomiques que l'on rencontre quand l'on doit développer sur du tactile.

Mots-clés : tactile, Microsoft Surface, jeu, C♯

Summary

Imagine yourself in your living room, you're in family and your children want to play a board game. In the middle of the room between your TV and your couch is a table, it seems perfectly normal...however, you hit with your finger is then start Microsoft Surface, a touchscreen latest technology. It offers a multitude of applications, you can play chess, watch your photos, etc.. your whole family gathers around the technology. With the touch computing is more individual, but shared.

This project is to create a game for the table touchscreen it's a puzzle game called Tangram. In this report you will see among other technologies that allowed me to create this game such as the C♯ and ergonomic problems that occur when we must build on the touchscreen.

Key-words : touchscreen, Microsoft Surface, game, C♯