GRANIER Myriam
VEYSSIERE Célian

Licence WEB

Création d'un jeu de SOKOBAN en C#

Institut Universitaire de Technologie de Clermont Ferrand

année 2007-2008

Remerciements

Nous tenons à remercier notre tuteur de projet M. CHASSAGNE, qui nous a suivi lors de la réalisation du projet. Nous remercions également M. SALVA, responsable de la licence web, ainsi que les autres professeurs et élèves qui nous ont soutenus pendant toute la durée du projet.

Sommaire

Introduction

Lors de notre Licence Professionnelle option Développeur d'Applications Intranet/Internet à l'Institut Universitaire de Technologie (IUT) de Clermont-Ferrand, nous avons eu à réaliser un projet tutoré.

Le projet en question consistait en la réalisation d'un jeu en C# -le Sokoban. Celui-ci fait partie de notre apprentissage dans le domaine des smart applications.

Tout d'abord, nous présenterons notre contexte de travail. Ensuite, nous verrons en détail le sujet. Puis, nous nous pencherons sur l'analyse de celui-ci. Nous poursuivrons avec la réalisation du projet. Enfin, nous traiterons des résultats et des perspectives.

1 - Contexte de travail

1.1) Institut Universitaire de Technologie

L'Institut Universitaire de Technologie (IUT) de Clermont-Ferrand propose une Licence Professionnelle option Développeur d'Applications Intranet/Internet. Dans le cadre de cette licence, nous avons eu à réaliser, par binôme, un projet tutoré.

1.2) Projet tutoré

Le projet tutoré a pour but de permettre aux étudiants d'améliorer leurs compétences et leur capacité à travailler en groupe. Il prépare au milieu professionnel, et aux projets de type dit 'industriel', c'est-à-dire destiné à un client. La réalisation de ce projet se fait en un temps limité de six mois.

Notre groupe est tutoré par M. CHASSAGNE, professeur intervenant de .Net dans la licence. Le projet qu'il nous a confié consiste en la réalisation d'un jeu de Sokoban, célèbre casse-tête, en C#.

1.3) Besoin et sujet

Le but ici est de réaliser un jeu de casse-tête 'smart client' consistant à diriger un personnage qui doit déplacer des caisses de manière à résoudre un puzzle.

L'interface doit être esthétique et intuitive, et le jeu accessible à divers publics avec un niveau de difficulté variable. Il sera plus amplement défini dans la partie suivante de notre rapport.

Afin de répondre à ce besoin, il était nécessaire de voir les différentes phases de la programmation. Il nous a donc fallu, dans un premier temps, comprendre le fonctionnement du jeu, et chercher des approches pour aboutir à un plan de réalisation. La documentation sur le jeu, la réalisation du cahier des charges et l'analyse ont été des éléments nécessaires au projet nous permettant de mieux comprendre et maîtriser notre objectif.

Dans un second temps, nous avons entrepris la programmation à proprement parler. Celle-ci découlait directement de l'analyse. Elle a été découpée en modules réalisés en fonction de leur importance et testé au fur et à mesure afin d'aboutir à un jeu fonctionnel.

2 - Sujet détaillé

2.1) Définition des termes du sujet

Tout d'abord, nous allons faire un petit rappel de ce qu'est le **SOKOBAN**. Il s'agit d'un jeu vidéo de résolution de puzzle ayant été inventé au Japon (*où le nom s'écrit* 倉庫番 *ou sôkoban*). Le jeu original a été écrit par Hiroyuki Imabayashi et comportait 50 niveaux. Depuis, il existe des dizaines de versions différentes avec toujours davantage de niveaux.

Le principe du jeu en 2D est simple : gardien d'entrepôt, le joueur doit ranger des caisses sur des cases cibles, l'entrepôt étant divisé en cases. Nous vous invitons à consulter manuel et cahier des charges pour en savoir plus.

Ensuite, nous allons définir le C#. Il s'agit d'un langage de programmation orienté objet à typage fort (il garantit que les types de données employés décrivent correctement les données manipulées). Il a été créé par Microsoft spécifiquement pour la plateforme .NET, dont il est de fait dépendant. Il est proche du JAVA, bien que des similitudes le rendent davantage semblable au C et C++.

2.2) Notions essentielles

Afin de comprendre la démarche entreprise pour la réalisation de notre projet, mais aussi de rendre possible la compréhension de sa structure pour une reprise future éventuelle de notre application par un autre programmeur il convient de comprendre certaines notions essentielles que nous allons détailler ici.

2.2.1) Framework .NET

Qu'est ce que la plateforme Microsoft .NET ?

Il s'agit d'un ensemble de produits et de technologies de l'entreprise Microsoft. La plupart dépendent du « Framework .NET » (« espace de travail ») qui est un composant d'exploitation Windows constituant un équivalent de machine virtuelle. Ce Framework a été conçu par Anders Hejlsberg, père du Delphi. Il est téléchargeable sur le site de Microsoft.

Son point fort réside dans le fait que l'on peut choisir le langage que l'on désire utiliser : C#, J#, VB.NET, etc. Il a pour but de faciliter la tâche des développeurs, notamment ceux qui maîtrisent la Programmation Orientée Objet, car la plateforme propose une hiérarchie d'objets ainsi qu'une harmonisation générale des API. Cela simplifie le déploiement et la maintenance d'applications.

Il s'agit d'un environnement dit « managé » ; c'est-à-dire qu'il gère les aspects suivants :

- il alloue la mémoire pour le stockage des données et des instructions du programme.
- il gère (autorise ou interdit) les droits de l'application.
- il démarre et gère l'exécution.
- il gère la réallocation de la mémoire pour les ressources qui ne sont plus utilisées.

Il existe également une version « light » du Framework sous Windows: le *.NET Compact Framework* ainsi que des implémentations libres de .NET telles que Mono et DotGNU.

2.2.2) SMART Client

En informatique, et selon la définition donnée par Microsoft, un smart client (« client intelligent ») désigne, de manière générale, une application que l'on peut facilement déployer et gérer, tout en ayant accès à une expérience interactive riche. La capacité d'adaptation et la rapidité de réponse sont assurées par l'utilisation des ressources locales et des connexions établies de manière intelligente avec des données réparties (sur un serveur, une autre machine, …) quand il y en a.

Cela offre au développeur l'avantage d'un accès complet à l'ordinateur et au réseau (utile par exemple dans le cas d'une option multi joueurs). De plus, il n'y aura pas besoin de gérer un maintien de session puisque tout sera déjà sur le client. En revanche, il est important de noter que l'application est non portable, c'est-à-dire qu'étant développé pour Windows, elle ne fonctionnera que sous ce système d'exploitation. Enfin, on peut également préciser que ce type d'application nécessite une installation sur le client avant de pouvoir être utilisée.

2.3) Logiciels utilisés

Etant donné que le sujet impose le C# comme langage de programmation et un jeu en réalisation, le choix et l'utilisation de logiciels se trouve limitée. Toutefois, nous pensons utile de préciser les outils qui auront servi à sa conception.

Afin de réaliser notre projet, il a été décidé d'utiliser le logiciel Visual Studio 2005 qui est celui par défaut pour traiter le C# et celui sur lequel nous avons été formés, il est donc en notre possession, contrairement à la version 2008.

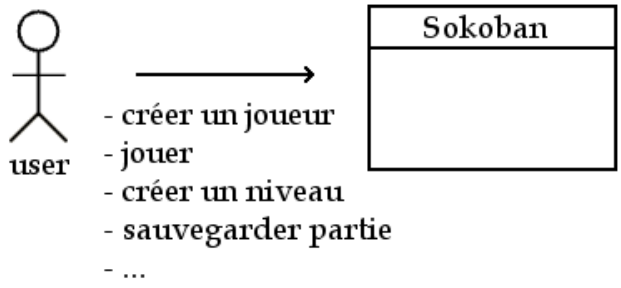
Concernant la partie « graphisme » de l'application, les éléments nécessaires à l'affichage des niveaux seront réalisés sous Paint Shop Pro puisque nous savons déjà comment nous en servir et que l'application gère les formats d'images qui nous intéressent.

Maintenant que le travail à faire est cerné, nous allons pouvoir entreprendre son traitement, en commençant par l'analyse qui permettra entre autre de dégager le détail de chacune des fonctionnalités voulues et de s'organiser.

3 - Traitement du sujet : Analyse

3.1) Diagrammes UML et morcellement du travail

3.1.1) Diagramme de contexte

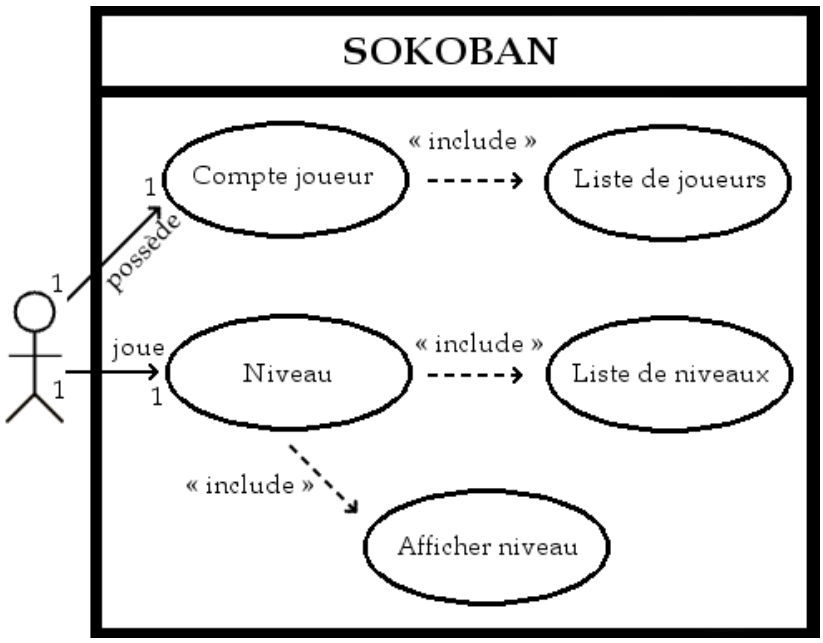


Il n'y a qu'un seul acteur principal « user » qui sera le joueur.

Celui-ci interagira avec le système que sera le jeu Sokoban, notamment par l'intermédiaire du menu, lui offrant la possibilité de jouer, créer un niveau, etc.

Ces actions sont pour le moment externes au système, en revanche, les traitements qui suivent ces actions lui seront internes.

3.1.2) Diagrammes des Cas d'utilisation

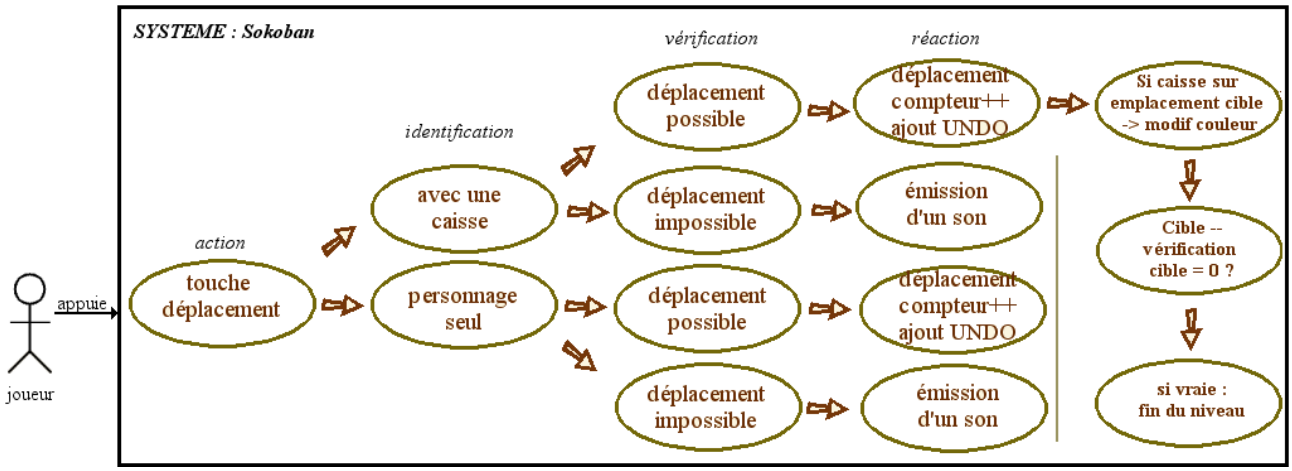


Le joueur possède un compte qui est inclus dans une liste de joueurs.

Il peut jouer sur un niveau contenu dans une liste de niveaux. Un niveau étant passé par un module d'affichage pour rendre la partie possible pour le joueur humain.

Détaillons une des actions offertes par le menu, par exemple une fois la partie lancée (avec un joueur identifié), l'utilisateur a la possibilité de jouer en dirigeant son personnage.

Chaque déplacement va engendrer un traitement :

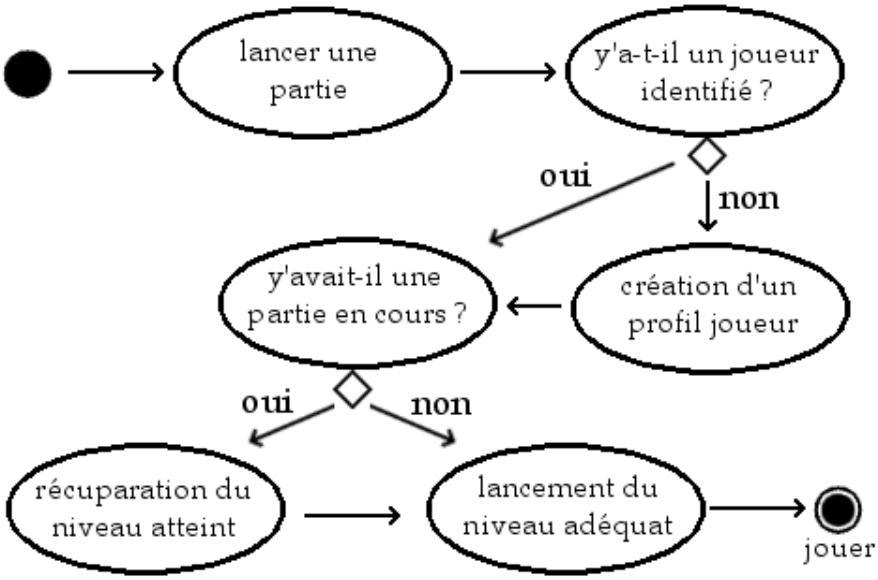


Le programme va identifier si le personnage est seul ou si une caisse se trouve sur son chemin, ensuite il va vérifier si le déplacement est possible ou non. S'il ne l'est pas, le personnage ne bougera pas, et on pourra par exemple émettre un son signalant à l'utilisateur que son coup est refusé. Si, en revanche, le déplacement est possible, il sera effectué.

Cela entraîne un certain nombre de réactions : incrémentation du compteur de pas, ajout du déplacement effectué en mémoire (UNDO), vérification de l'emplacement de la caisse. Si cette dernière est sur un emplacement cible il faut décrémenter le compteur 'cible' comptabilisant le nombre de caisses restant à placer. Si elle sort d'un emplacement cible, il faut le ré-incrémenter, et sinon on ne fait rien. Dans le cas de la décrémentation de 'cible' il faudra vérifier s'il est à zéro ou non. S'il est à zéro alors il n'y a plus de caisses à placer, et le niveau est fini.

3.1.3) Diagramme d'activité

Partons d'une situation initiale où, par exemple, le joueur demande à lancer une partie.



Le programme va devoir vérifier si cette demande a été effectuée par un joueur identifié ou pas. Si tel n'est pas le cas, il demandera la création d'un profil de joueur. Ensuite, il va interroger la base de données pour savoir si oui ou non, une partie était en cours avec ce profil. Si oui, il ira chercher la partie du joueur en cours (récupération du ou des niveaux qu'il a débloqué). Une fois cette vérification faite, le joueur pourra choisir, dans la liste de niveaux disponibles, celui auquel il désire jouer, et ce niveau sera lancé pour arriver à un statut final où l'utilisateur pourra jouer.

A savoir que le schéma est sensiblement simplifié vu qu'à la création d'un joueur, il y a vérification de l'unicité du nom.

Maintenant que nous savons ce que seront les fonctionnalités de base pour le jeu, il faut découper le travail à effectuer en plusieurs parties, afin de pouvoir commencer par développer celles qui seront essentielles à l'application, ainsi qu'évaluer celles qui s'articuleront tout autour.

3.1.4) Découpage en différentes parties

Le jeu doit être réalisé sous forme d'applications qui sera divisée en plusieurs parties plus ou moins distinctes :

Tout d'abord un module principal, minimum requis pour avoir un jeu fonctionnel, qui comportera:

- Un menu principal rassemblant les principaux points d'accès vers les différentes parties du programme. Ce menu sera complété à chaque ajout de module secondaire.
- Un écran de connexion donnant la possibilité de créer ou supprimer des comptes utilisateurs, ainsi que d'activer un de ceux-ci pour pouvoir jouer.
- Un panneau de configuration permettant de conserver les informations de personnalisation de l'application.
- Un écran de jeu permettant de jouer au Sokoban.
- Un écran permettant de visualiser les meilleurs scores.

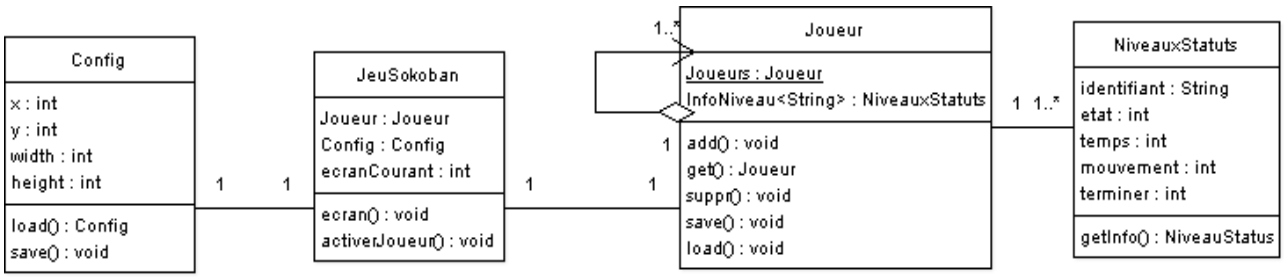
Ensuite des modules secondaires offrant de nouvelles fonctionnalités telles que:

- Un écran permettant d'avoir une vue rapide sur un ensemble de niveaux et de choisir parmi ceux-ci
- Un écran de connexion pour pouvoir jouer à plusieurs.
- Un écran de création de niveaux.

3.1.5) Détail de la configuration et du menu

Le jeu possède une classe de Configuration contenant les informations sur le logiciel qui seront ou non utilisées par le programme suivant les choix de l'utilisateur.

Une liste de joueurs est ensuite récupérée via la classe Joueur contenant une liste de Joueurs avec leurs informations.



Pour une navigation plus aisée dans le programme une barre de menu sera affichée en permanence avec, une fois tous les modules développés, l'arborescence suivante :

Jeu	**Paramètres**	**?**
□ Menu Principal	Configuration	□Aide de jeu
□ Nouvelle partie / Continuer		□À Propos
□ Joueur		
□ Quitter		

La partie « jeu » contient les éléments permettant de jouer ou d'arrêter de jouer.

Menu Principal : Renvoi au menu principal d'où que l'on soit.

Nouvelle partie : Renvoi à un début de jeu

Joueur : Permet d'accéder à la partie permettant de changer, créer ou supprimer des joueurs

Quitter : pour quitter l'application.

La partie « configuration » permet juste d'accéder au panneau de configuration général de l'application.

Le partie « ? » se concentre sur l'aide et l'apport d'informations sur ce jeu.

Aide de jeu : affiche un écran avec l'aide jeu : explication et règles

À Propos : affiche des informations sur l'application : auteurs, version, etc.

3.2) Détail des écrans

3.2.1) Changements des panneaux

Le programme est conçu autour d'un socle commun avec une *barre de menu*, contenant les chemins d'accès rapide dans le programme et un *panel* permettant l'affichage des panneaux de l'application.

Pour simplifier leur changement, ces panneaux n'étant pas très nombreux, ils sont identifiés par une constante.

La classe principale a le rôle de retenir les informations sur le joueur courant, le panneau courant ainsi que le précédent et le suivant pour la navigation.

Le premier panneau chargé est celui du « menu principal », sans aucun utilisateur créé ou chargé (sauf si cela est demandé via le panneau de configuration).

3.2.2) Menu Principal

Celui-ci comporte deux états : connecté et déconnecté.

En mode déconnecté certains boutons envoient directement au panneau d'identification avant de pouvoir aller plus loin.

Ce panneau donne la possibilité d'aller dans les parties suivantes en modifiant le panel. Pour ce faire, on passe par les boutons suivants :

- Nouvelle partie : qui donne la possibilité de commencer le jeu suivant un niveau de difficulté choisi par le joueur.
- Continuer : qui lance la carte de niveau listant tout ceux finis ou accessibles.
- Créer ou changer de joueur : qui affiche le panneau d'identification.
- Partie multi joueur : qui permet de lancer un panneau pour chercher des utilisateurs sur le LAN pour jouer à plusieurs.
- Créer un niveau : lance l'éditeur de niveau du jeu.
- Quitter : pour quitter l'application en sauvant les paramètres (suivant la configuration).

3.2.3) Panneau de configuration

Ce panneau offre :

- La possibilité de garder les positions et taille de la fenêtre pour les restaurer à la réouverture de l'application.
- La possibilité de demander à ce que le dernier joueur soit pré chargé au lancement.

Les informations seront enregistrées dans le fichier XML de configuration via l'objet de configuration.

3.2.4) Panneau d'identification

Ce panneau joue le rôle de gestionnaire des utilisateurs. Ceux-ci y sont gérés de la façon la plus simple, seul l'identifiant permet de les distinguer. Ce dernier devra être unique. En cas de doublon trouvé, seul le premier sera reconnu. La liste de joueur est chargée au démarrage de l'application. Celle-ci est stockée dans le dossier /Saves/ avec un fichier par joueur. Cela permettra de rendre les informations des joueurs transportables.

Ce panneau permet d'accéder à:

- une liste de joueurs, avec la possibilité de les activer ou les supprimer (avec confirmation).
- un champ de saisie pour entrer un nouveau joueur (qui sera ajouté dans la liste s'il est valide).
- un label donnant le nom du joueur qui est actuellement identifié par le programme.

Un bouton permettra de se rendre sur le panneau suivant qu'il s'agisse de « difficulté », « création de niveau », ou « mutli-joueurs » (, si ces modules ont tous été développés).

3.2.5) Panneau de difficulté

Celui-ci contient une liste de lanceur :

- facile
- moyen
- difficile
- niveaux de joueurs

Les trois premiers lancent le premier niveau du jeu de la difficulté sélectionnée. Le dernier, faisant partie d'un module secondaire, lance la carte de niveaux créés par les joueurs.

3.2.6) Panneau de la carte des niveaux

Le panneau se divise en 2 parties :

- Une partie affiche les miniatures des niveaux d'une sélection de niveaux. Chaque zone a un écouteur permettant de savoir si elles sont cliquées ou survolées.

- Au survol, les informations des niveaux sont affichées dans la seconde partie. Au clic, il en sera de même, mais le niveau choisi sera activé et prêt à être lancé. Cela désactive également le niveau précédemment sélectionné, s'il y en avait un.

Un bouton permet de lancer le niveau sectionné (bouton valide uniquement si un niveau a été sélectionné).

3.2.7) Panneau d'un niveau

Le panneau contient une zone d'information comportant:

- - Temps,
- - Distance parcourue (nombre de coups utilisés),
- - Nombre de caisses restantes à ranger,
- - La possibilité de faire « undo », « redo » ou recommencer.

Le undo revient en arrière dans les déplacements, et redo ré effectue le dernier mouvement annulé. Ces deux options font partie d'un module secondaire qui sera intégré au jeu si le temps imparti le permet.

Il y a aussi une zone d'affichage du niveau dans laquelle se déroulera le jeu.

3.2.8) Panneau de connexion multi joueurs

Ce panneau permettra de trouver les autres joueurs sur un réseau local et de pouvoir lancer une partie de 2 à 4 sur le même niveau. Chacun a son niveau propre sur son écran et donc son propre jeu avec ses déplacements et le nombre de blocs à ranger. Le temps est commun.

Le choix d'un niveau se fait par le joueur qui a lancé la partie, via la carte des niveaux.

Ce panneau est intégralement considéré comme un module secondaire qui sera développé en tout dernier lieu si le temps imparti le permet.

3.3) Le jeu

3.3.1) Affichage

L'affichage se fait par blocs contenant chacun une image représentant l'élément de décor qu'il incarne.

Les changements d'états se feront en fonction des modifications effectuées sur le tableau en deux dimensions représentant le niveau.

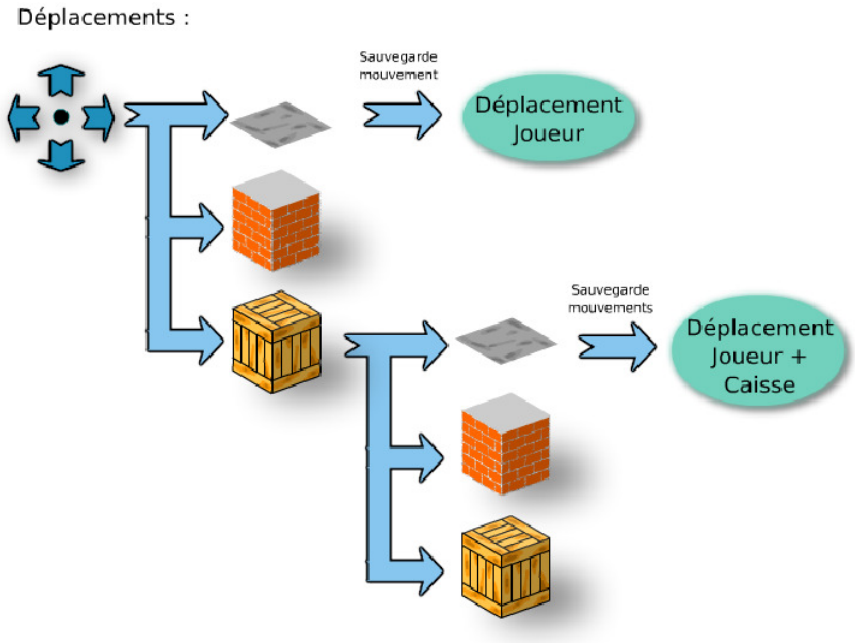
Le personnage est affiché sur les zones de déplacements autorisées que sont le sol neutre et les zones de rangement des caisses.

3.3.2) Gestion du clavier

Les événements sont envoyés à une classe qui sera répercutée sur la table de représention du niveau (l'affichage).

En fonction des règles établies dans le jeu, le déplacement sera effectué ou non.

Un déplacement n'est autorisé que s'il n'y a rien sur la case cible, où qu'il y a une caisse et rien derrière celle-ci. Toute autre action n'est pas acceptée.



3.3.3) Fin d'un niveau

Un niveau est achevé lorsque toutes les boites sont rangées. Suite à cela, le niveau est enregistré comme 'fini' pour le joueur, si ce n'est pas déjà le cas. Son score aussi est enregistré, dans le tableau des scores.

Si le niveau fini appartient au jeu de base, le niveau suivant est également débloqué et cela est enregistré dans les informations du joueur.

Une fois l'analyse terminée, il ne reste plus qu'à la mettre en application en vue de réaliser le projet.

4 - Réalisation

4.1) Gestion des déplacements

La base du jeu repose sur le fait que le personnage se déplace dans l'entrepôt, il faut donc repérer les déplacements, et les gérer.

Les fonctions qui suivent permettent de récupérer les évènements de clavier. Suivant la touche pressée, on exécute la fonction de déplacement correspondante et on réoriente le personnage grâce à la fonction ci après :

```
protected override bool ProcessCmdKey(ref Message msg, Keys keyData)
        {
            switch (keyData)
            {
                case Keys.Right:
                    deplacement(1, 0);
                    personnage.ImageLocation = (appPath + @"\Images\perso_droite.png");
                    return true;
                case Keys.Left:
                    deplacement(-1, 0);
                    personnage.ImageLocation = (appPath + @"\Images\perso_gauche.png");
                    return true;
                case Keys.Up:
                    deplacement(0, -1);
                    personnage.ImageLocation = (appPath + @"\Images\perso_haut.png");
                    return true;
                case Keys.Down:
                    deplacement(0, 1);
                    personnage.ImageLocation = (appPath + @"\Images\perso_bas.png");
                    return true;
                default: return base.ProcessCmdKey(ref msg, keyData);
            }
        }

    protected override void OnKeyDown(KeyEventArgs e)
    {
        base.OnKeyDown(e);
    }
```

Celle-ci vérifie notamment ce qu'est la touche (« Key ») sur laquelle on appuie. Si c'est la flèche droite (« Key.Right ») alors le personnage se tourne vers la droite.

Vient ensuite le problème de gérer le personnage qui bouge et les caisses. La fonction qui permet leur déplacement est la suivante :

```
public void deplacement(int x, int y)
{
      // test si le déplacement est possible (détection de mur ou de caisse)
      if (testDeplacement(x, y))
      {
            // si le déplacement est possible, il regarde s' il y a une caisse devant
            if (testCaisse(x, y))
            {
                  //  On récupère l'identifiant de la caisse
                  int id = niveauElements[_ypos + y, _xpos + x];
                  //  On modifie la position de la caisse à l'écran
                   caisses[id].Location = new System.Drawing.Point((x*2 + _xpos) * _xpas,
(y*2 + _ypos) * _ypas);
                  //  On teste pour savoir si la caisse est placée sur 1 zone de stockage
                   if (testCaissePlacee(x*2, y*2))
                   {
                         // Changement d'état de la caisse : celle-ci est placée
                         caisses[id].ImageLocation =
                               (appPath + @"\Images\caisse_placee.png");
                         etatCaisse[id-1] = 1;
                   }
                  else
                  {
                        // Changement d'état de la caisse : celle-ci n'est pas placée
                         caisses[id].ImageLocation = (appPath + @"\Images\caisse.png");
                         etatCaisse[id-1] = 0;
                  }
                  // Test pour savoir si le niveau est fini
                  testFin();
                  // Mise à jour de la grille :
                  niveauElements[_ypos + 2 * y, _xpos + 2 * x] =
                                    niveauElements[_ypos + y, _xpos + x];
                   niveauElements[_ypos + y, _xpos + x] = 0;
            }
            // la dernière position est ajoutée au undo
            undoAdd(_xpos, _ypos);
           // Modification de la position du joueur :
            _xpos += x;
            _ypos += y;
            _x += x * _xpas;
            _y += y * _ypas;
            // Le compteur de mouvement est incrémenté
             mouvements++;
            // On déplace le personnage à l'écran suivant :
            personnage.Location = new System.Drawing.Point(_x, _y);
      }
}
```

x est le déplacement horizontal (soit 1, 0, -1),

y est le déplacement vertical (soit 1, 0, -1).

Il faudra bien évidemment prendre en compte qu'un déplacement peut entraîner la fin du niveau lorsque toutes les caisses auront été placées sur leur cible.

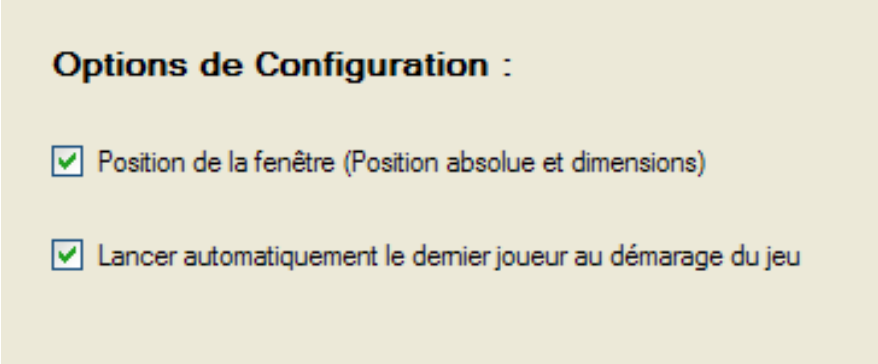
Pour vérifier quand cela arrive, on utilise cette fonction ci :

```
public void testFin()
{
	// compte le nombre de caisses placées
	int compteur = 0;
	for (int i = 0; i <  nombreCaisse; i++)
	{
		compteur += etatCaisse[i];
	}
	// si le nombre de caisses placées est identique au nombre de caisses totales, on
lance la procédure d'achèvement du niveau :
	fin = (compteur == nombreCaisse) ? true : false;
	if (fin)
	{
		niveauAcheve();
	}
}
```

Nous avons donc là déjà le déplacement du personnage et la fin du niveau. Nous allons nous intéresser à la gestion des utilisateurs ou « joueurs » que l'application devra gérer.

4.2) Gestion des utilisateurs et Configuration

La configuration de l'application (cf. image ci après) devra être mémorisée. Sa taille et sa position, ainsi qu'éventuellement le précédent joueur en cours d'utilisation, devront être restaurés à chaque démarrage du jeu.

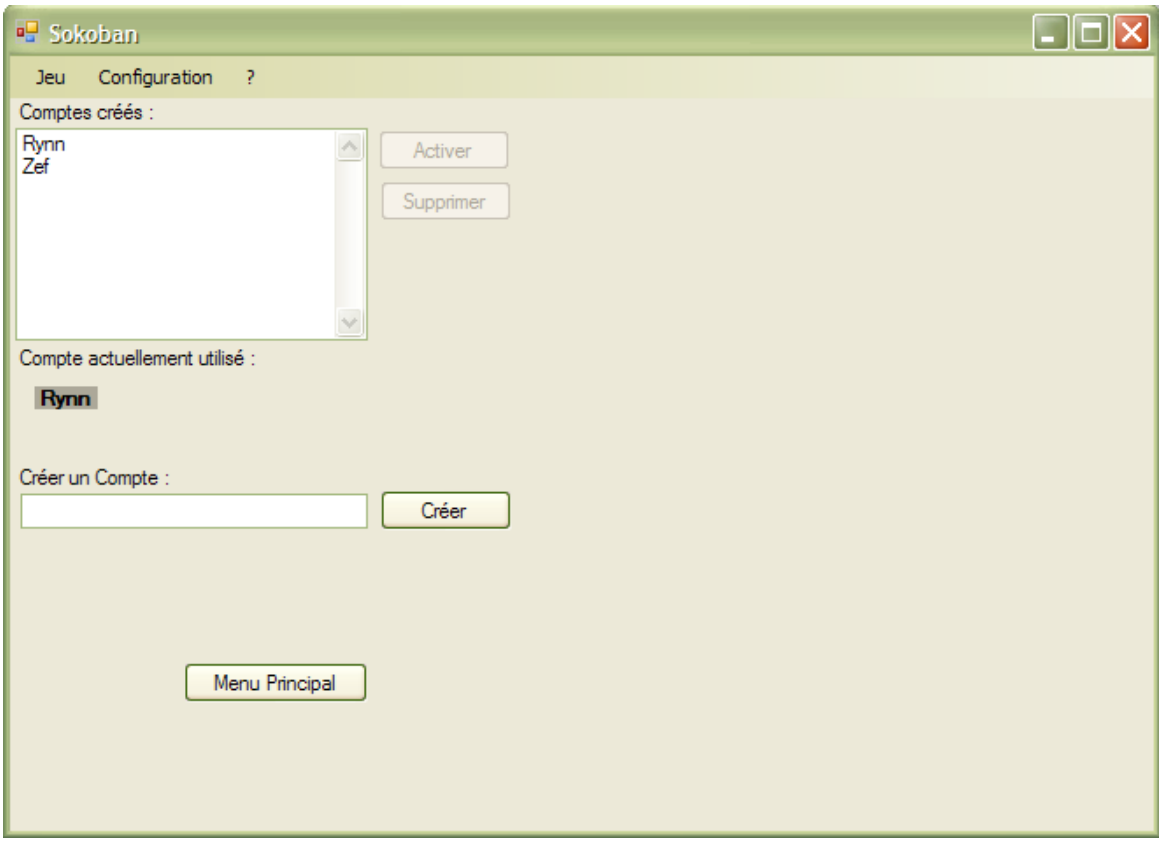


De même, il devra être possible d'avoir plusieurs joueurs, sur la même machine, qui seront en mesure de jouer chacun leur partie. Ces joueurs et leur avancée doivent donc être toujours conservés de la fermeture à la réouverture de l'application.

Les classes « Joueur » et « Config » devront donc être `[Serializable]`. Les joueurs et leur avancée seront enregistrés dans un fichier du style « nomDuJoueur.Xml » qui sera lu lors de la récupération des données.

Au lancement de l'application, quand l'option a été cochée dans la configuration, le dernier joueur activé est rechargé. Cette option est activée par défaut.

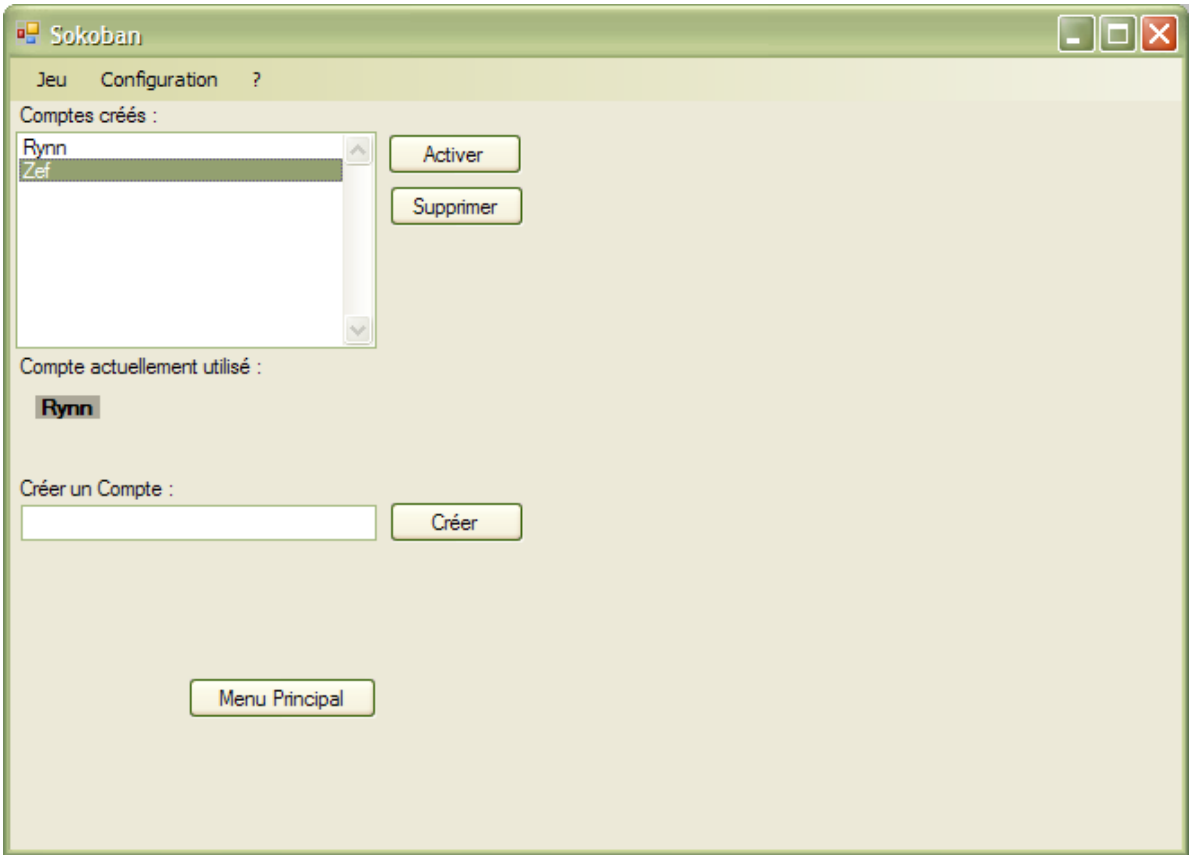
On peut constater dans la page identification que son nom apparaît comme « actuellement utilisé » :



Il est possible de retourner au menu principal pour jouer avec le compte activé, de créer un nouveau compte, ou d'en activer un existant qui soit autre que celui déjà en cours d'utilisation.

Pour se faire, il suffit de sélectionner dans la liste le compte voulu, le bouton « Activer » devient alors disponible, de même que le bouton supprimer, pour le cas où l'on veuille détruire un compte qui ne servirait plus.

Par exemple ici, le joueur Zef est sélectionné. On peut choisir de l'activer afin de jouer avec au lieu d'utiliser le compte Rynn, ou le supprimer :



Le bouton « Menu Principal » permet, comme son nom l'indique, le retour au menu de l'application. Il est également accessible par l'onglet « *Jeu > Menu Principal* ».

Pour créer un compte il suffit de compléter le champ prévu à cet effet et de cliquer sur le bouton « Créer ». Si le nom est déjà pris, un message d'erreur s'affichera, sinon, le joueur sera créé. Il apparaîtra dans la liste avec les autres comptes et pourra être utilisé.

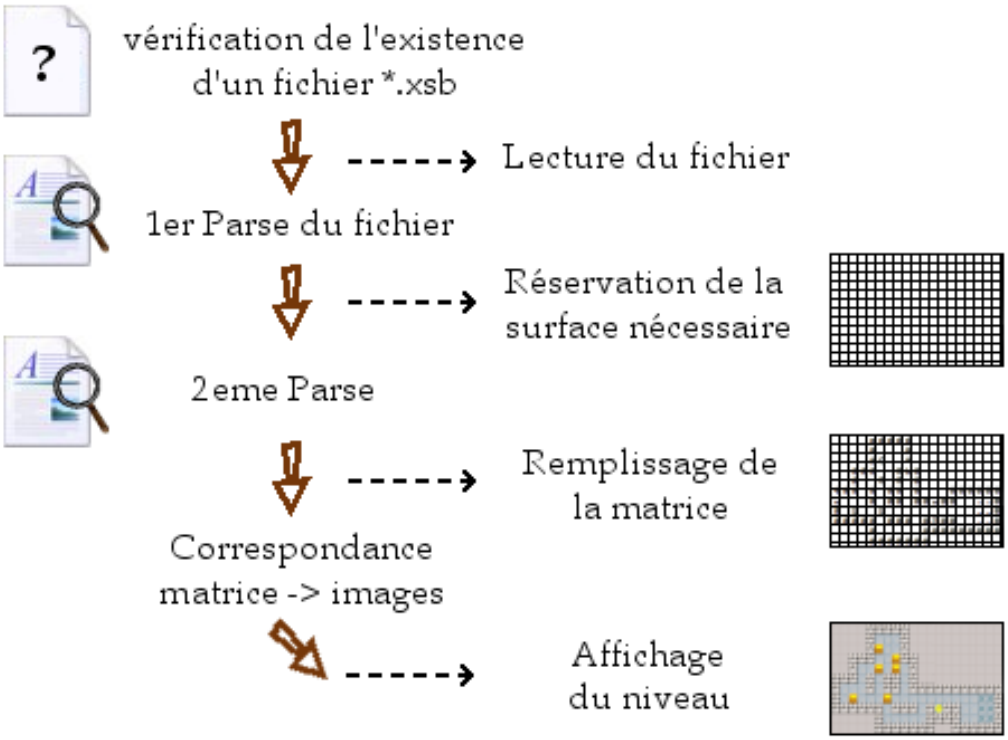
4.3) Les niveaux

Les niveaux sont des *.xsb qui sont lus par l'application et générés ensuite sous forme graphique.

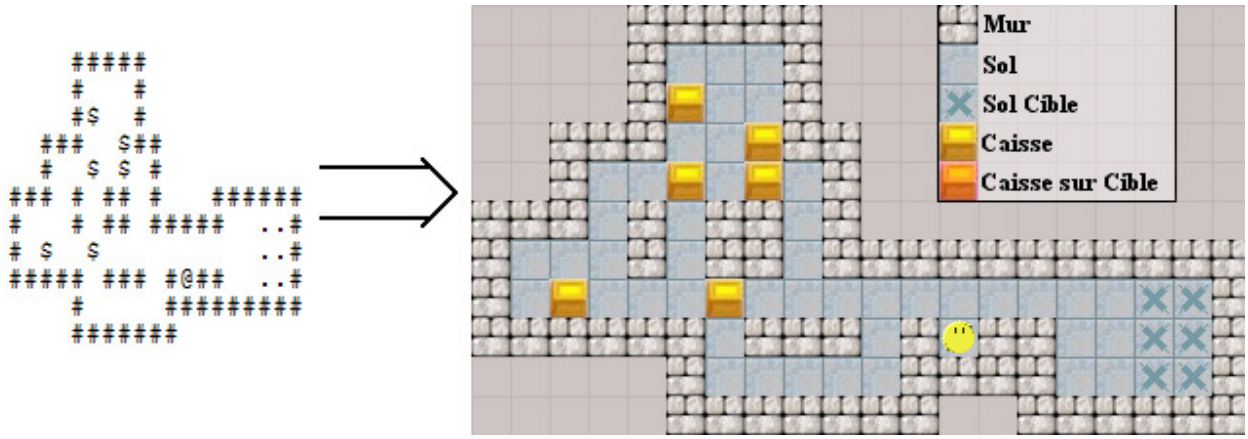
Pour se faire, il passe deux fois par le parseur. La première fois, il alloue la surface, en nombre de lignes et de colonnes, dont il aura besoin pour représenter le niveau, créant ainsi deux matrices : une « fixe », contenant les cibles et les murs, et une « vivante » pour les caisses et le personnage qui sont appelés à se déplacer. Le personnage est lui placé à une position matricielle sans être inscrit dans ces deux matrices. Il se déplace en fonction de celles-ci.

La seconde fois, il remplit chaque matrice avec les caractères correspondants à chaque case. Ensuite ces matrices sont lues et les caractères sont changés en images de manière à pouvoir afficher le niveau avec le graphisme.

Si l'on schématise de manière simplifiée ce traitement, cela donne ceci :



Par exemple, le niveau que nous avions présenté dans le cahier des charges va devenir :



Ici, le bonhomme jaune est le personnage que l'on déplacera. Il n'y a aucune caisse en rouge (placée) car aucune n'est sur un emplacement cible, signalé par une croix. Il est évident qu'il y a autant d'emplacements cibles que de caisses.

Mur et caisses (si elles sont devant un mur ou une autre caisse) seront des éléments bloquants. Le personnage est confiné dans l'entrepôt.

5 - Résultats et perspectives

5.1) Résultats

Au final, nous avons développé une application fonctionnelle possédant la moitié des objectifs de départ.

Ainsi, le joueur aura la possibilité de

- jouer aux niveaux qui seront lus par l'application,
- sauvegarder des paramètres propres à l'application,
- sauvegarder son ou ses profils (étant conservés, il sera possible de trouver plusieurs joueurs sur une même machine).

Nous n'avons pu achever que cette partie du sujet qui nous avait été donné étant donné qu'il manque les modules secondaires suivants :

- des niveaux, le minimum de 15 n'ayant pas été atteint, bien qu'il suffise de les trouver au format .xsl et de les faire lire par l'application pour les ajouter,
- le redimensionnement du panel de « jeu » n'est pas géré,
- la sauvegarde est automatique et non pas au choix du joueur,
- le module du mode « multi joueur » est absent.

Toutefois, les modules développés fonctionnent parfaitement et rendent l'utilisation de l'application tout à fait possible même en l'absence des autres modules.

5.2) Difficultés

Au cours de ce projet, nous avons rencontré des difficultés. Elles sont diverses et nous n'avons hélas pas pu toutes les surmonter.

Celles que nous avons rencontrées au cours de l'analyse se portent essentiellement sur l'utilisation de l'UML car nous n'en avions que très peu fait auparavant. La mise en forme d'une analyse, et la visualisation des éléments lui étant nécessaires n'étant déjà pas simple, il nous a fallu effectuer de nombreuses recherches, et en discuter. Ces difficultés se sont également trouvées au niveau de l'organisation et du travail en équipe. Habitués à travailler seuls, il nous a fallu apprendre à mettre en commun le travail, à s'adapter à la vision de l'autre, à communiquer plus volontiers les résultats… Ce qui nous a pris plus longtemps qu'on ne l'aurait cru. Cette mise en route difficile nous a beaucoup ralenti.

D'autres difficultés se sont posées au niveau de la programmation qui, se faisant dans un langage encore relativement peu connu, nous a demandé encore une fois un fort travail de recherche et de réflexion.

La dernière difficulté étant le temps qui passe hélas bien vite et qui également bien plus complexe à gérer.

Ces difficultés nous auront néanmoins beaucoup apporté. Nous avons appris à travailler en binôme, à puiser l'information plus facilement, ou en tout cas plus efficacement, et à organiser notre travail. Nous avons également gagné des connaissances en C#.

Nous regrettons beaucoup de n'avoir pas pu développer tous les modules, ou en tout cas une plus grande partie, que ce qui nous a été possible de faire, ce qui nous amène aux perspectives d'évolution de notre application.

5.3) Perspectives

Notre projet n'ayant pas été parfaitement terminé du fait de l'absence de certains des modules initialement prévus, il va de soi que les premières améliorations envisageables à lui apporter seraient leur développement, à savoir :

- un module Multi joueur permettant à plusieurs personnes (de 2 à 4) de s'affronter sur un niveau, choisi aléatoirement ou par le joueur lançant la partie, du Sokoban.
- la possibilité de laisser au joueur le choix du type de sauvegarde.
- gérer le redimensionnement du panel de jeu,
- gérer la création de niveaux par l'utilisateur.

Il serait aussi envisageable de rajouter des modules qui n'ont pas été évoqués dans le sujet de départ et qui pourraient être « divertissants » pour l'utilisateur tels que :

- un module de choix de design : l'utilisateur pourrait modifier le graphisme des niveaux, notamment à leur création, afin de varier leurs décors et éléments.
- un module permettant au joueur de créer ses propres niveaux à la main et de les tester, que nous avions envisagé mais pour lequel le temps nous a manqué.

En conclusion, nous pouvons dire que même si l'application n'a pas pu être finalisée comme nous l'aurions souhaité, nous sommes satisfaits de ce qui a pu être réalisé et qui fonctionne.

Conclusion

Ce projet nous aura beaucoup apporté notamment au niveau des compétences servant à la vie professionnelle.

En effet, celui-ci nous a offert d'apprendre à travailler en groupe, chose sans laquelle notre projet n'aurait jamais pu être réalisé, car même s'il est incomplet, il n'en demeure pas moins fonctionnel.

Nous avons appris à nous organiser et à résoudre les problèmes rencontrés par nous même.

Ce projet est également le premier projet destiné à un utilisateur que nous développons. Le fait de devoir se mettre à la place de l'utilisateur afin de mieux cerner les besoins et les méthodes pour y répondre requiert d'envisager les choses sous un aspect différent de celui dont nous avons l'habitude.

Il nous aura aussi permis de mettre en application, et de développer, les acquis que nous avions, ainsi que d'acquérir de nouvelles compétences comme par exemple au niveau du C# ou de l'UML que nous connaissions peu.

Au final, ce projet aura été très intéressant et nous aurons malgré tout pris plaisir à le développer.

Annexes

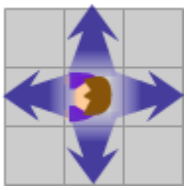
Le Sokoban

- Mini Manuel -

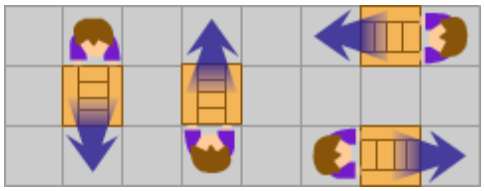
- SOMMAIRE -

Avec le contrôle d'un personnage, il faut apporter des caisses des points de départ à un lieu de rangement.

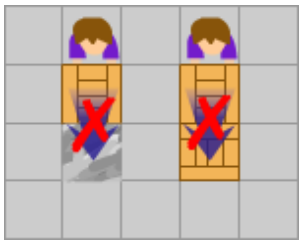
Le personnage ne peut se déplacer qu'en suivant les cases verticalement ou horizontalement :



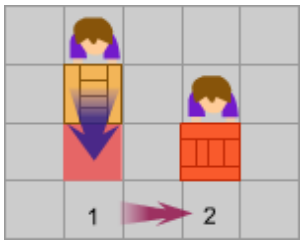
Le personnage peut pousser les caisses uniquement et une seule à la fois. Il ne peut pas les tirer.



Les murs sont des éléments bloquants, tout comme les caisses. Il est impossible de pousser deux casses à la fois car cela est bien trop lourd.



Une fois la caisse posée dans une des zones de rangement, celle-ci change de couleur. Il est toujours possible de la déplacer (mais elle reprendra sa couleur d'origine si elle quitte l'emplacement cible).



Les zones sont généralement fermées. Il s'agit de trouver la bonne solution pour déplacer les caisses jusqu'à leur emplacement d'arrivée en prenant en compte ces contraintes. Les niveaux sont créés de façon à rendre la chose plus ou moins ardue. Bien entendu, cette difficulté sera au début pédagogique, puis au fil des niveaux, cela demandera un degré de réflexion au joueur toujours plus élevé.

1.2 Les niveaux

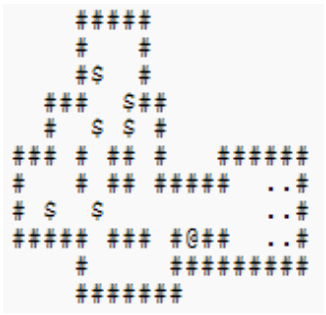
Un niveau réussi se solde par la possibilité de se rendre au niveau suivant (logique), mais également la possibilité de pouvoir refaire le niveau réussi autant de fois que le joueur le désire.
[Certaines versions du Sokoban permettent d'accéder à tous les niveaux sans avoir réussi les niveaux précédents, ou d'avoir trois niveaux possibles en plus de ceux réussis (Exemple, si les niveaux 1, 2 et 4 sont réussis, je peux me rendre aux niveaux 3, 5 et 6).]

1.3 L'éditeur de niveaux

La possibilité de créer des niveaux était déjà présente dans les premières versions Sokoban. Pour créer des niveau de façon simple (et accessible partout) ceux si sont réalisés dans un fichier texte qu'un simple éditeur de texte permet d'éditer.

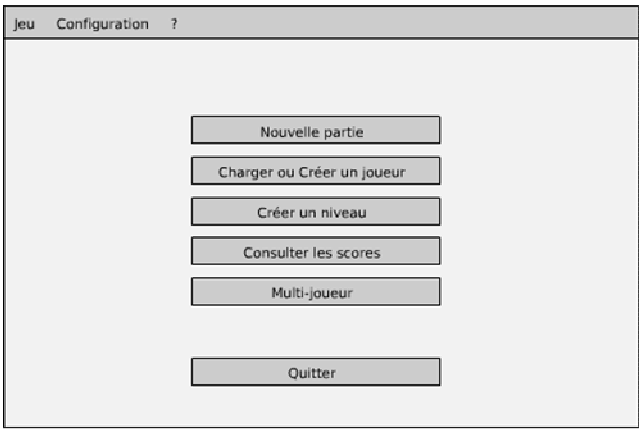
#	Murs
$	Caisses
.	Zone de rangement
*	Caisse sur une zone de rangement
@	Personnage
+	Personnage sur une zone de rangement

Créer un niveau est simple. Mais le rendre jouable est plus complexe, mais ceci est à la charge de celui qui crée les niveaux. Il pourra, via l'interface du jeu, tester à sa volonté sa création.
Voici un exemple de niveau en code :



2 - Navigation dans l'application

Le menu principal permet d'accéder à diverses sections :



- Nouvelle partie
- Charger/Changer ou Créer un joueur
- Créer un niveau
- Consulter les scores
- Multi joueur

2.1 Nouvelle partie

Lance la carte de niveau, si aucun joueur n'est sélectionné ou existant dans la base, il demandera si il doit en créer ou en reprendre un avant de débuter le jeu.
La carte des niveaux permet d'accéder au(x) niveau(x) disponible(s) par défaut mais aussi aux niveaux créés par le joueur.
Les deux pages, « normal » et « création », possèdent la liste complète de niveaux marqués dans des petites cases. Le survol de chaque case permet de voir une miniature du niveau ainsi que son degré de difficulté et quelques informations sur ceux-ci.
Un bouton sera prévu pour revenir au menu principal.

2.2 Charger/Changer/Créer un joueur

Cette partie permet de revenir ou changer de profil pour pouvoir reprendre une partie en cours. Il sera aussi possible de créer un nouveau joueur avec un profil vierge.
Il sera également possible d'annuler et revenir au menu principal.

2.3 Créer un niveau

Ici, un joueur aura la possibilité de créer et tester lui-même ses niveaux.

2.4 Consulter les scores

Cette rubrique donnera l'occasion à l'utilisateur de consulter ses scores par niveaux.
En fonction de l'avancée de la programmation, il sera peut être possible d'accéder également au TOP des scores tous joueurs confondus.

2.5 Multi joueur

Le mode multi joueur permet de s'affronter à plusieurs sur les mêmes niveaux.

Le Sokoban

- Cahier des Charges -

1. Introduction

1.1. Equipe autour du projet

L'équipe se constitue de trois personnes :

- M. CHASSAGNE Frédéric (fchassagne@deltamu.fr), Tuteur de projet,
- Mlle GRANIER Myriam (myriam-h.granier@laposte.fr), développeur,
- M. VEYSSIERE Célian (zefling@yahoo.fr), développeur.

1.2. Environnement

Ce projet de jeu entre dans le contexte des Projets de Licence Professionnelle de Développement d'Application Internet et Intranet de l' Institut Universitaire de Technologie de Clermont-Ferrand.

2. Description de la demande

2.1. Objectifs

Ce jeu doit être composé de :

* Création d'un ou plusieurs joueurs.
* Plusieurs niveaux pour jouer.
* Plusieurs niveaux de difficulté.
* Classement par temps possible.
* Possibilité de redimensionner la fenêtre.
* Sauvegarde des paramètres activables.
* Sauvegarde du jeu avec choix de la méthode.
* Un mode multi joueur.

2.2. Description générale

Le jeu de sokoban est un casse-tête où vous dirigez un personnage qui doit placer des caisses à des emplacements cibles uniquement en les poussant une à une.

2.3. Fonctionnalités

* Le jeu doit être capable de garder en mémoire plusieurs joueurs ayant chacun leur pseudo, leur avancée, leurs niveaux crées, leurs scores par niveau et par difficultés.

* Il possèdera un nombre déterminé de niveau à résoudre (minimum 15) de difficulté croissante.

* Il sera possible de déterminer le niveau global de difficulté : simple / normal / difficile, du jeu. Cela jouera principalement sur les options disponibles (nombre de coups, temps, undo limité, etc.…)

* Il devra exister au moins un classement : le meilleur sera celui qui résoudra un niveau en un temps le plus court possible.
Possibilité d'ajouter un classement en fonction du nombre de coups utilisés également.

* La fenêtre du jeu sera redimensionnable selon le souhait de l'utilisateur.

* L'utilisateur pourra choisir dans les options, en cochant une case, de conserver les paramètres de tailles de la fenêtre afin de les ré appliquer lors du prochain lancement de l'application.

* Un utilisateur/joueur pourra choisir la méthode de sauvegarde de sa partie (Xml/Base de données).

* Il sera possible de jouer au SOKOBAN en réseau, de 2 à 4 joueurs.

Choix 1 : La fenêtre sera alors en plein écran le temps de la partie, séparée en 4 parties, chacune présentant la fenêtre d'un des joueurs.

Choix 2 : (plus vraisemblable) Chaque joueur ne verra que sa propre fenêtre, lorsque le premier a fini, le résultat est affiché.

Possibilité de mettre un choix « Fin » pour terminer le jeu, ou « Continuer » pour permettre aux trois autres joueurs de se départager.

2.4. Réception

Ce jeu est destiné dans un premier temps à M. CHASSAGNE, chef de projet, puis dans un deuxième temps, à tous ceux qui désireront s'en servir comme divertissement.

Il sera considéré que les objectifs devant être remplis par le produit sont atteints si :

* le jeu fonctionne sans erreur.
* le jeu offre une interface agréable.
* toutes les fonctionnalités devant être intégrées au produit ont bien été développées.

3. Contraintes

3.1. Langage

Ce projet s'inscrivant dans un cadre universitaire afin d'apprendre à maîtriser le C#, ce dernier est donc le langage imposé pour la conception.

3.2. Esthétisme

Le projet étant un jeu, l'interface jouera un grand rôle. De fait, elle devra être soignée et agréable.

3.3. Temps

Etant donné que ce projet s'inscrit comme Projet de Licence, il a été donné une date butoir d'achèvement, à savoir avant les stages qui débutent fin mars début avril.

4. Déroulement du projet

4.1 Planification

Voici les différentes étapes au cours desquelles sera développé le produit :

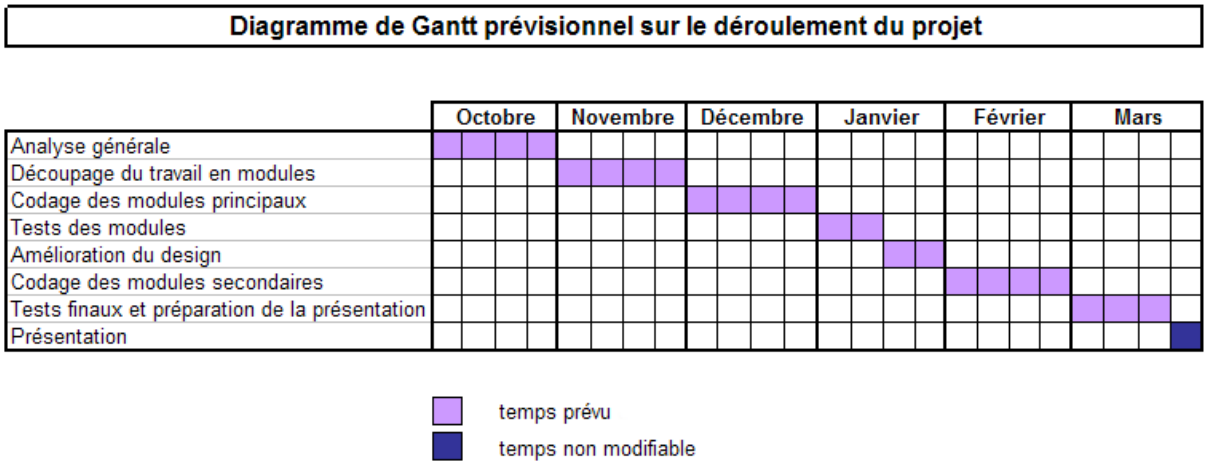
* Analyse
    - Rédaction d'un cahier des charges
    - Elaboration des cas d'utilisations
    - Schématisation des éléments intervenants
    - Schématisation des mécanismes du jeu
* Programmation
    - Codage de modules les uns après les autres
    - Tests de l'application, des modules
    - Mise en forme de l'interface
* Finalisation
    - Ajout de modules complémentaires
    - Amélioration du design
    - Test par un utilisateur
    - Correction de bugs éventuels

4.2 Ressources

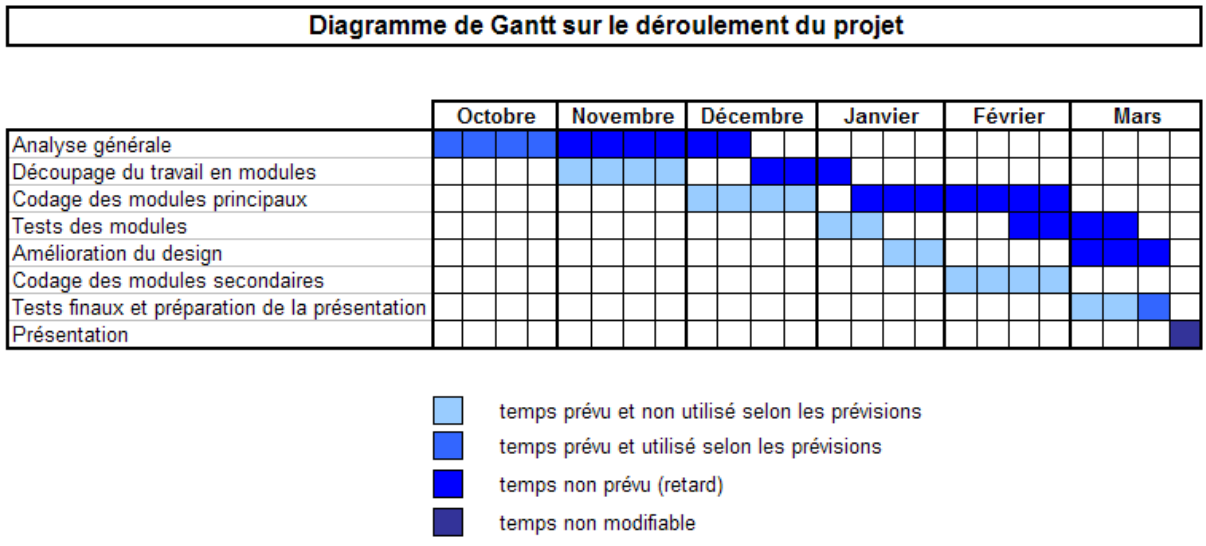
Sont mises à notre disposition toutes les ressources nécessaires à la réalisation du projet à savoir livres, Internet, etc…

Diagrammes de Gantt

1. Diagramme prévisionnel :



2. Diagramme réel :



English Summary

During our Data-Processing License, the realization of a project has been entrusted to us. This project was actually to develop a game named Sokoban in order to offer an entertainment to the user, and to apply our knowledge of the C#, which was the language we had to use.

In a first time, we analyzed the subject in detail and separated the different tasks to do in several modules according to their importance. It takes use more time than we think, especially because we didn't know how to do correctly an analysis.

In a second time, we began to program the application with Visual Studio 2005. We first developed the more important module and then, following the time we had, some secondary modules.

In a last time, we tested our application, and tried to improve its design to make it pleasant for the user.

We encountered some problems on the work in group, making a correct analysis, or doing our project in time; but we managed to finish the major part of what we had to do.

Finally, our game is functional even if we couldn't finish it like we wanted to. This project was interesting to realise and it had permit us to learn many things like working together with other people, take of the information we need, organize our time on a task, etc.