GConfs - TP «Bien commencer son projet de sup»

Mickael *Mogmi* Bidon (bidon_m) Pierre-Marie *PMdomine* de Rodat (de-rod_p)
Adrien *Cookieslover* Conrath (conrat_a) Julien *Sn00ze92* Lehuen (lehuen_j)

27 novembre 2009

Table des matières

1 Introduction **3**
1.1 Quelques liens utiles . . . 3

2 Présentation du jeu : Sokoban **4**

3 Analyse préliminaire **6**
3.1 Le travail d'équipe . . . 6
3.2 Déroulement de l'analyse . . . 6
3.3 Définition des types de données . . . 7
3.4 Découpage du projet . . . 8

4 Bonus **10**
4.1 Le score . . . 10
4.2 Des graphismes sayksys . . . 10
4.3 Un menu . . . 10
4.4 Un éditeur de niveaux . . . 10
4.5 It's not a bug, it'a a feature ! . . . 10
4.6 Comprendre la blague de l'image de la page de garde . . . 10

5 Conclusion **11**

1 Introduction

Bienvenue au très attendu… TP *Bien commencer son projet* !

Nous avons organisé ce TP dans l'espoir de vous donner un petit aperçu de ce qu'est la réalisation d'un projet comme celui que vous avez à rendre à la fin de l'année, mais en miniature. En effet, une nuit devrait être nécessaire pour réaliser un petit jeu, ce qui vous donnera une expérience en programmation.

Pour coller un minimum avec la réalité, nous nous limiterons à vous donner des indications sur la démarche à suivre et la structure globale du programme : il n'y aura pas de code à trou ou d'autres choses dans le genre. Mais nous restons à votre disposition pour toute question, chose pas claire, … Et bien sûr, vous programmerez avec le langage que vous avez choisi pour votre projet : OCaml, C# ou F#.

De même, nous ne vous obligeons à utiliser aucune bibliothèque externe. À vous de choisir ce dont vous avez besoin… comme dans votre véritable projet ! Nous vous proposons SDL pour sa simplicité (et parce qu'on peut l'utiliser en OCaml et F#), mais rien n'empêche les groupes utilisant C# d'utiliser XNA. Quelque soit votre choix, il vous faudra de toute façon lire et comprendre la documentation… comme dans votre véritable projet !

Concernant l'utilisation d'un gestionnaire de version, elle aurait semblé logique à partir du moment où on travaille en groupe. Malheureusement, nous n'avons pas à notre disposition de serveur proposant 80 comptes pour le TP ; l'échange des sources se fera donc par dossier `public` interposé ! L'avantage est que vous apprécierez réellement lors de votre projet de pouvoir en utiliser un ! Plus sérieusement, le TP est organisé de telle manière à ce que les « conflits » entre vos sources soient minimisés.

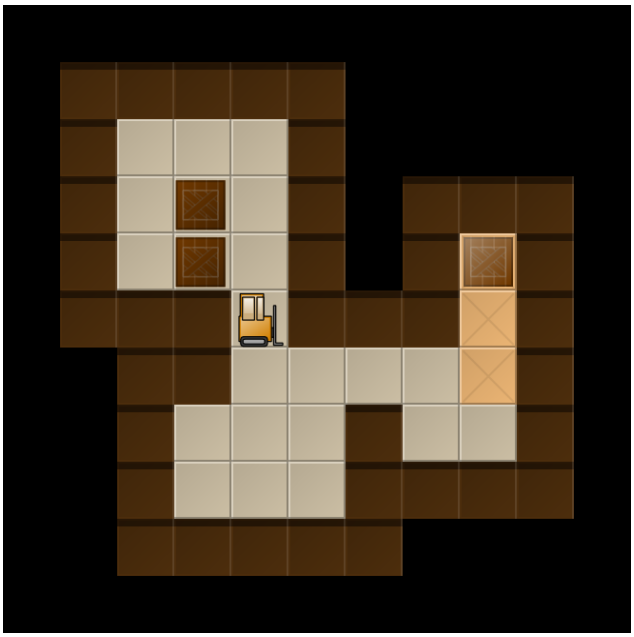
Bon travail !

1.1 Quelques liens utiles

Le nécessaire du C# http://fr.wikibooks.org/wiki/Programmation_C_sharp
(en) C# Station http://www.csharp-station.com/tutorial.aspx
(fr) DA-OCAML http://www.pps.jussieu.fr/Livres/ora/DA-OCAML/
(en) Riemer's XNA Tutorial http://www.riemers.net/
(fr) Tutoriel SDL http://ln-s.net/4d+D

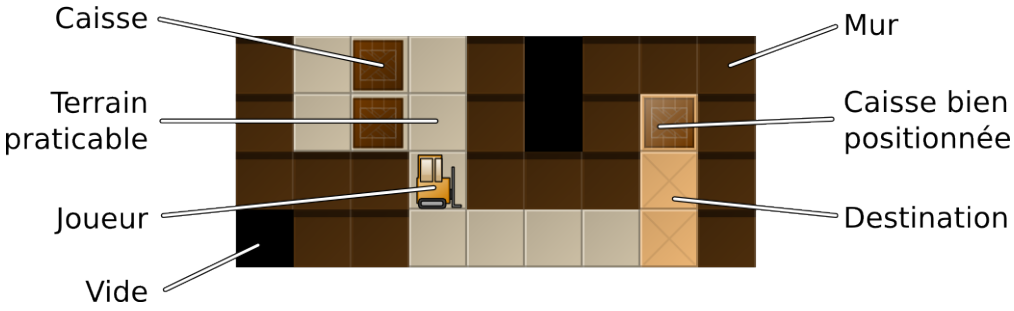
2 Présentation du jeu : Sokoban

Nous avons beaucoup hésité sur le type de jeu que nous allions vous proposer (notamment entre un Tank, un Asteroïd et un Sokoban). Mais puisque pour la plupart d'entre vous, la programmation est quelque chose de tout nouveau, nous avons opté pour un sujet simple, et toutefois intéressant ! :-)

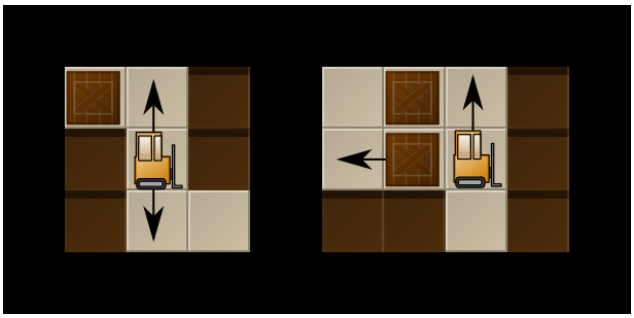


Des fois que certains ne connaitraient pas ce jeu, je vous invite à googler sans plus attendre :-) ! (`http://fr.wikipedia.org/wiki/Sokoban` pour les flemmards).

Le principe est plus que simple : ce n'est pas dessus que vous butterez. Le Sokoban met en scène un seul joueur dans un envirronement en grille 2D (qui est différente selon les niveaux). Le terrain (chaque case) est soit praticable (c'est le sol), soit non praticable (mur ou vide).



Des caisses (qui prennent naturellement une case chacune) sont présentes, et le but du jeu est de les amener dans des emplacements bien précis dépendant du niveau. Le personnage ne peut se déplacer qu'horizontalement et verticalement par cases adjascentes, pas diagonalement. Pour déplacer une caisse, il lui suffit de « pousser » dessus… et pour que le déplacement soit valide, il faut donc que la caisse puisse elle-même se déplacer dans la direction voulue.



Le joueur a gagné quand toutes les caisses sont positionnées sur les emplacements.

3 Analyse préliminaire

3.1 Le travail d'équipe

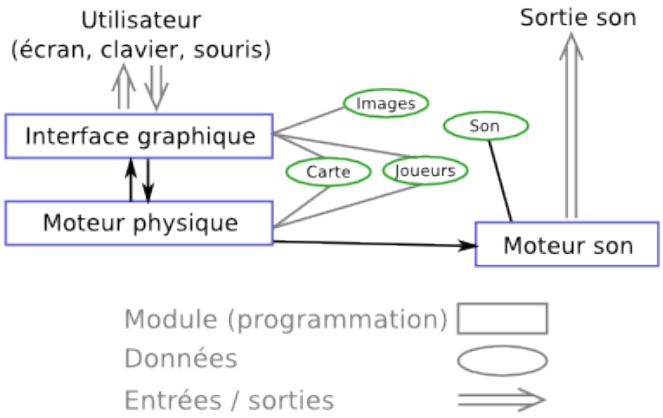
En premier lieu, quelle est la différence majeure entre un *gros TP* et un projet ? Elle réside principalement dans le fait que cette fois vous n'êtes pas seul pour réaliser votre projet ! Vous êtes généralement par groupes de quatres personnes. C'est à la fois un avantage conséquent, puisque les différentes tâches du projet peuvent alors être répartie. Cependant cela provoque également l'apparition de difficultés nouvelles : besoin de de coordination, se mettre d'accord, ne pas empieter sur le travail des autres etc.

En principe pour ce TP vous êtes venus en groupe, c'est donc un excellent moment pour commencer à voir comment fonctionne le travail d'équipe ! Ne négligez surtout pas cet aspect.

3.2 Déroulement de l'analyse

La première chose à faire avant de se lancer dans un projet, même la réalisation d'un petit jeu comme celui que nous vous proposons, est de réfléchir au découpage, à l'organisation de votre projet. Ici commence donc le processus qui va vous amener à déterminer comment vous aller organiser votre code. C'est une étape très importante qui vous fera perdre du temps par la suite si elle est bâclée ! Cela correspond à l'approche « du global vers le détail » (ou *top-down* pour les pseudo-scientifiques) vue pendant la conférence.

Tout d'abord… notre jeu, le Sokoban dans son intégralité… qu'est-ce qu'il contient, déjà ? Comment le structurer ? Pour le coup, ce n'est pas sorcier : pratiquement tous les jeux ont le même schéma.



Les données représentent tout ce qui n'est pas du code. « Le code » représente ici ce qui fait quelque chose, ce qui agit, et donc les instructions. Les données sont par opposition les types des variables ainsi que les variables elles-mêmes. On peut ainsi voir un programme comme une machine qui reçoit des données, les traite, et retournent d'autres données… un peu comme les routines des langages (les fonctions et les procédures).

Ansi, avant de se lancer dans le code, et avant même de penser au code, il faut penser aux données que va manipuler le code ! C'est la base commune (le code de tout le monde travaille sur les données), et il ne fait pas la louper. Pour les données, et seulement pour les données, nous allons un peu vous guider, en explicant les choix que nous avons fait. Il faudrait faire cette unité une fois par groupe (tous devant un PC) pour ensuite l'utiliser comme base de travail, chaque membre du groupe commençant sa partie.

3.3 Définition des types de données

On peut déduire du sujet que le programme devra manipuler une grille limitée pour représenter le terrain : chaque case représente un genre de terrain. Ah, du coup il faut aussi définir un type qui représente le genre de terrain ! Résumons déjà ce que nous avons :

- Un type pour le genre de terrain : `TCaseType`, c'est une énumération (un type somme en OCaml) de : `Out` (le vide), `Floor` (le sol, on peut se déplacer dessus), `Wall` (un mur), `Destination` (l'emplacement où le joueur doit placer les caisses).
- Un type pour la grille du terrain : une classe `Grid` (un enregistrement en OCaml) regroupant les dimensions de la grille (largeur et hauteur, `Width` et `Height`) et un tableau `Grid` à deux dimensions devrait suffire.
- En prévision, on peut créer un type pour représenter les quatre directions `Direction` : `Left`, `Top`, `Right`, `Bottom`.

Détaillons la classe `Grid` (en OCaml, on créer un type enregistrement et on définit de simples fonctions à la place des méthodes) :

- Le constructeur prend en paramètre deux entiers `W` et `H`. Il initialise le tableau aux bonnes dimensions, renseignes les données de la grille en général.
- Une méthode `Get` qui prend en paramètre deux entiers `X` et `Y` et qui retourne la valeur de la case correspondante. Il faut gérer le cas où les coordonnées désignent une case « en dehors » de la grille : si `X` ou `Y` est négatif, par exemple, on retourne `CaseType.Out` (donc toute case hors de la grille est considérée comme du vide).
- En complément de `Get`, la méthode `Set` prend en paramètre deux entiers `X` et `Y` et un type de case (`CaseType`) pour modifier la case de la grille désignée. Bien sûr, avant de faire la modification, il faut vérifier que les coordonnées soient valides.

C'est bon, les « moules » (comme dans « moulage », bien sûr) pour créer un terrain sont prêts ! Passons maintenant aux objets.

Nous aurions pu intégrer la notion de caisse aux différents types de terrain (dans le type `CaseType`), mais cela aurait un peu compliqué les choses : il aurait fallu distinguer le cas où une caisse est sur le sol et le cas où une caisse est sur une case de destination. Une autre manière de voir les choses est de repérer les caisses par leurs coordonnées uniquement. À ce moment-là, l'ensemble des caisses est un simple tableau de coordonnées.

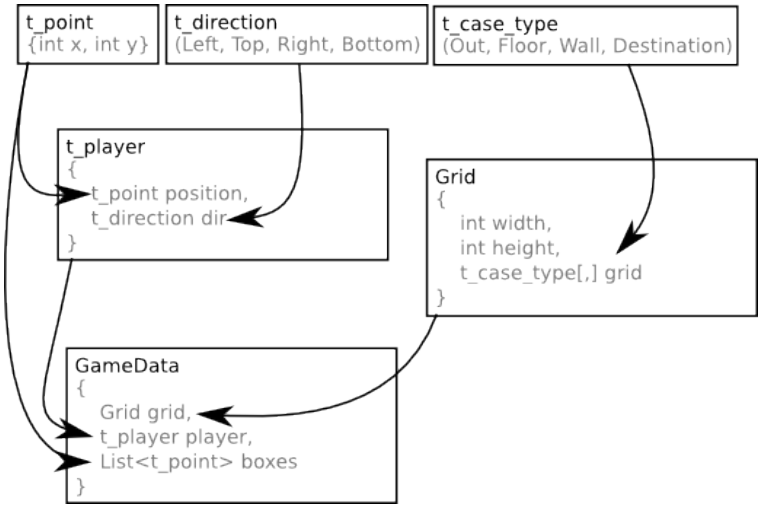
Enfin, le personnage est aussi un objet qui se déplace, mais on peut aussi mémoriser son orientation : il regarde soit vers la gauche, le haut, ...

Pour les types des objets, on a donc :

- Le type `Point` est une simple structure (un enregistrement) de deux entiers : `x` et `y`.
- Le type `Boxes` est une liste de caisses.
- Le type `Player` est une autre structure regroupant : `Pos` de type `Point` (c'est la position du joueur) et `Dir` de type `Direction` (c'est la direction vers laquelle le personnage regarde.

On termine cette définition de types communs avec la classe `GameData`. Celle-ci sert uniquement à regrouper les différents éléments du jeu, c'est une question de pratique. Cette classe regroupe :

- la grille de type `Grid`
- le joueur de type `Player`
- la liste de caisses `Boxes`



C'est tout ce qui concerne les types avec lesquels tout le groupe va travailler. L'étape suivant est donc de savoir qui travaille sur quoi !

3.4 Découpage du projet

Chaque partie « métier » du projet est censée remplir *une seule* tâche bien précise, tandis que des parties « glu » servent à utiliser ensemble les partiers métier. Dans ce projet simple, seule une partie « glu » est nécessaire : elle utilise tous les moteurs pour faire fonctionner le jeu.

Le but du découpage est donc de structurer le programme, mais aussi à permettre la répartition des tâches ! Chaque personne doit s'occuper d'un moteur, mais tout le monde doit avoir auparavant décrit comment les différentes parties s'utilisent, par exemple : comment le moteur graphique va-t-il chercher une case à afficher dans le moteur physique ? C'est une partie très importante pour avoir des parties qui se complètent et s'utilisent facilement lorsque le code est « relié ».

Le *parser* À l'origine de tout… il y a la lecture des niveaux. Le parser s'en occupe : on lui donne un nom de fichier. Il ouvre alors le fichier désigné, le lit pour déchiffrer son contenu et retourne une grille ! L'utilisation du parseur est donc assez simple : une seule routine doit être connue par les autres.

Le moteur physique Dans cette partie du projet, vous devez définir tout ce qui concerne la *physique* de votre jeu. Dans un premier temps, le moteur physique charge un niveau avec le parser. Il propose ensuite plusieurs routines représentant des actions : déplacer le personnage, déplacer des caisses, détecter la fin de partie, … tout en vérifiant à chaque fois si c'est valide ou non ! Parmi ces vérifications : est-ce que le personnage peut se déplacer dans telle direction ? Et pour une caisse ?

Le moteur graphique Le moteur graphique s'occupe de deux choses : d'une part, il doit afficher la partie en cours à l'utilisateur, donc afficher le terrain, les caisses et le personnage. D'autre part, il s'occupe de recevoir les « entrées utilisateur » (c'est-à-dire les événements du clavier, de la souris, la fermeture de fenêtre, …) et de retransmettre ces demandes d'actions au moteur physique.

Le moteur de son Bien que non obligatoire, le moteur de son représente tout de même un élément important de votre futur projet (son âme, toussa toussa, sauf que vous allez piquer l'âme d'un autre jeu sans

doute :D). Il reçoit des demandes de la part du moteur physique. (c'est donc le moteur physique qui appelle le moteur de son)

Les autres Rien ne vous empêche de créer un second moteur graphique pour créer une interface de jeu (menus, configuration, ...), un module le meilleurs score et autres ! Voir les bonus pour ça...

Comme vous l'avez sûrement remarqué, les différentes parties dépendent d'autres... on peut alors se demander comment faire pour développer : doit-on attendre que le gars qui s'occupe du parser aie fini pour commencer le moteur physique ? Bien sûr que non !

Il s'agit en fait de faire une sorte de remplaçant temporaire : à l'endroit où on devait utiliser le parser pour charger un niveau, on remplit manuellement (dans le code) un petit niveau pour les tests, et on peut commencer à coder le moteur physique ! Il suffira ensuite de remplacer ce morceau de code lorsque le parser sera prêt !

Un petit exemple :

```
/* Code à écrire si le parser est fonctionnel. */
Grid grille = new Parser().Parse("mon_fichier.txt");

/* Code à écrire si le parser n'est pas encore disponible. */
CaseType out = CaseType.Out;
CaseType wall = CaseType.Wall
CaseType destination = CaseType.Destination;
CaseType floor = CaseType.Floor;
Grid grille = new Grid(4, 4);
grille.Grid =
{
  {out, wall, wall, wall},
  {wall, floor, destination, floor},
  {wall, floor, floor, floor},
  {wall, floor, floor, wall}
};
```

Ces « trucs » sont possibles parce qu'on a défini auparavant tous les types de données utilisés. Ansi, au lieu d'utiliser le code encore non fait, on simule le chargement de données par des données en dur.

Contrairement aux TP que vous avez toutes les semaines, c'est à vous, par groupe, de définir les différentes routines que vous allez créer. Comme vous l'avez lu, ça servira aux autres de support pour écrire leur code en le faisant adapté au vôtre. Tout ça peut paraître abstrait, mais lancez-vous dans votre partie, n'ayez pas peur de vous tromper : vous avez la nuit pour tout refaire ! Amusez-vous bien. ;-)

4 Bonus

Puisque nous la durée de ce TP dépend fortement de chaque groupe, et au vu du nombre de personnes qui seront présentes, nous vous proposons quelques bonus pour aller plus loin :

4.1 Le score

Relativement facile à implémenter et toujours pratique, vous pouvez calculer et afficher un score basé sur le temps, le nombre de déplacement et de manipulation de caisses pendant les parties ! Le défi devient alors de faire le moins de déplacement possible le plus rapidement.

4.2 Des graphismes sayksys

Bon cet aspect ne dépend pas trop de quelles bibliothèques vous utilisez... mais plutôt de vos talents de graphistes ! Par exemple, si vous utilisez la SDL et que vous ne vous êtes pas trop embêtés, vous avez des cases colorées en guise de niveau/murs/personnages. Pas très sexy tout ça. Essayez donc d'utiliser des images pour rendre votre jeu beaucoup plus sympathique. Si vous avez déjà commencé à partir d'images, tentez d'ajouter des effets visuels (changements de couleurs violents, explosions de la carte etc.). Si vous avez fait tout ça, vous pouvez maintenant passer à la 3D ! :-D

4.3 Un menu

Avec votre version actuelle, lorsque vous lancez le jeu, vous lancez toujours la même carte. Pourquoi ne pas proposer une simple interface à l'utilisateur, pour permettre de choisir la carte qu'il souhaite utiliser ?

4.4 Un éditeur de niveaux

Si votre parseur marche bien et respecte la norme, vous devriez pouvoir éditer et charger vos niveaux très simplement. Cependant, c'est plus sympathique de proposer une interface utilisable à la souris et permettant d'éditer en WYSIWYG (Google :-D) les niveaux.

4.5 It's not a bug, it'a a feature !

Vous aurez sans doute l'occasion de comprendre le sens de cette phrase lors du TP ! :-) De temps en temps, une erreur donne un résultat super fun, donc gardez le en tête et réutilisez le plus tard ! Vous pouvez aussi partir en cacahuette totale, proposer de faire exploser les caisses ou de transformer le jeu en un pacman avec les caisses mouvantes, rajouter des types de sols, jouer des sons funs etc.

4.6 Comprendre la blague de l'image de la page de garde

Attention, il n'a pas encoré été démontré que ce soit possible, donc n'y passez pas trop de temps ! Si vous voulez d'autres défis dans le genre, allez sur http://hmm-la-bd.eu/.

5 Conclusion