

Projets OpenGL

Michaël Hoste
(michael.hoste@swing.be)

Introduction

Ce document accompagne deux projets OpenGL : PonGL et ISokoban.

PonGL est un petit jeu qui n'utilise pas de techniques fort avancées mais qui est tout à fait jouable et complet (si on met à part la partie réseau qui est toujours en construction). ISokoban, lui, utilise certaines techniques plus avancées (par exemple pour le texturing).

Librairie "Text"

Avant de partir dans l'explication des deux projets, je juge utile de parler de la librairie Text. Je l'ai créée dans le cadre de PonGL et je l'ai réutilisée dans le ISokoban. Elle consiste en quelques fonctions que j'ai écrites dans le but d'afficher du texte en OpenGL. Comme vous le savez, si on veut écrire du texte en OpenGL, on est obligé de passer par des fonctions d'écriture de texte vectoriel dont le choix de la police est restreint, ce qui n'est pas toujours pratique ni joli. J'ai essayé de trouver des solutions sur le net mais à chaque fois il fallait dépendre d'une autre grosse surcouche (comme SDL) ou même d'un début de moteur 3D, ce qui ne m'intéressait pas.

J'ai donc décidé d'écrire une petite librairie qui permettrait l'affichage de n'importe quelle police de caractère, de couleur arbitraire. Après réflexion, la méthode la plus simple que j'ai trouvée est d'utiliser une texture contenant tous les caractères d'une police et de la charger en mémoire. Après, pour l'écriture d'un texte, on aligne n faces séparées de telle sorte à ce qu'il n'y ai pas trop d'espaces entre chaque caractères (pour éviter une écriture de type "courrier") et on applique sur celles-ci la texture en la paramétrant pour qu'elle s'étire de manière à n'afficher que le caractère désiré.

La texture contenant tous les caractères d'une police est créée grâce à un programme génial que l'on peut trouver ici :

<http://www.lmnop.com/bitmapfontbuilder/>

Elle est chargée dans le programme grâce à une librairie de chargement TGA créée par Nate Miller en 1999 et encore couramment utilisée en OpenGL.

L'astuce qui permet de choisir la couleur de la police consiste à utiliser uniquement la couche alpha de la texture. Grâce à ça, on définit uniquement quelle partie du caractère est visible et quelle partie devra être transparente. Pour le choix de la couleur, un glColor juste avant suffit.

PonGL : Concept

PonGL est le premier projet OpenGL que j'ai fait, il m'a permis de mieux comprendre le fonctionnement d'OpenGL et de sa surcouche Glut. J'étais parti avec l'idée de faire un petit Pong en 2D et j'ai développé l'idée petit à petit pour au final obtenir un résultat plutôt convaincant et un peu plus original que le concept initial (ajout de la 3D, système de points, jeu solo, niveaux de difficulté, etc.)

En conséquence, ce projet contient plusieurs "erreurs de débutants" ainsi qu'une mauvaise architecture du code. Il est assez mal documenté, mal structuré et vraiment brouillon. Je doute qu'un autre que moi arrive à s'y retrouver.

Malgré tout ça, il est entièrement jouable et je pense que le code est assez robuste. Après de multiples utilisations, je n'arrive plus à trouver un seul bug (du moins, sur les 3-4 PCs aux configs différentes sur lesquels je l'ai testé). Je vois donc ce projet comme quelque chose qui se suffit à lui-même mais qu'il ne faudra pas trop retravailler par la suite.

PonGL : Fonctionnement

Le fonctionnement de PonGL est assez simple, je vais le décrire sous forme de quelques questions-réponses :

Quelles sont les règles du jeu ?

Les règles du jeu sont très simples. Le jeu est composé d'une ou de deux palettes, de quelques murs et d'une balle. Vous contrôlez une palette et votre but ultime est de ne pas laisser passer la balle derrière votre palette tout en essayant de la faire passer derrière la palette de l'adversaire (pour le mode 2 joueurs) ou en essayant de faire le plus gros score possible (pour le mode 1 joueur).

Comment on compte les points ? Et c'est quoi ces jauges et ces "A" en haut de l'écran ?

Le système de point est un peu particulier... En fait, c'est un peu comme au volley (avant que les règles ne changent...) : Il faut avoir l'avantage pour marquer le point. On peut voir qu'on a l'avantage quand la caméra est proche de sa palette ou bien quand on a un "A" imprimé au dessus de l'écran à coté de sa jauge. Le gagnant sera celui qui aura sa jauge pleine... Donc, étant donné qu'on peut récupérer les points perdus, un match serré peut durer assez longtemps.

Avec quelles touches joue-t-on ?

Les touches utilisées pour diriger la palette de gauche en mode 1 joueur ou la palette de droite en mode 2 joueurs sur un PC sont "haut", "bas", "droite" et "gauche" (en fonction de la position de la caméra).

Les touches utilisées pour diriger la palette de gauche en mode deux joueurs sur un PC sont "z" (haut), "s" (bas), "q" (gauche) et "d" (droite) (en fonction de la position de la caméra).

Les touches supplémentaires sont "barre d'espace" pour lancer la balle (en mode manuel) et le classique "ALT+ENTER" pour passer d'un mode fenêtré à un mode

plein écran.

PonGL : Technique

Timer

Comme indiqué plus haut, ce projet contient des erreurs de débutant. La plus frappante techniquement est celle du timer. Celui-ci consiste à faire tourner un jeu à la même vitesse sur tous les ordinateurs. Mon erreur a été d'utiliser un "déplacement par frame" au lieu d'un "déplacement par temps". Ce qui revient à dire qu'à chaque affichage d'image, les objets se déplacent d'une même distance.

C'est un problème qui n'en est pas un quand on utilise toujours le programme sur la même machine. Mais dès qu'il est utilisé sur une machine plus puissante ou moins puissante, la vitesse du jeu change complètement.

La solution que j'ai trouvée pour rendre la vitesse du jeu identique sur un maximum de machines est de limiter l'affichage à 85 images/sec.

Le prix à payer pour ce choix peu judicieux est qu'un ordinateur peu puissant n'arrivera pas à faire tourner le jeu à 85 images/sec et que donc, la vitesse sera réduite. Une autre conséquence est que, pour que le jeu fonctionne correctement, la synchronisation verticale de la carte graphique doit soit être désactivée, soit activée avec un taux de rafraîchissement de plus de 85 hertz.

Je pense que malgré ces conditions, le jeu marche correctement sur 90% des ordinateurs et que pour encore quelques pourcents, un réglage dans les propriétés de la carte graphique suffit à fixer le problème.

Collisions

C'est sûrement la partie la plus délicate du projet, les collisions entre la balle et les palettes. Je ne m'attendais pas à une telle complexité pour ce que je considérais comme un détail. Il y a des tas de cas à prévoir et il faut bêta-tester souvent pour éviter des problèmes qui pourraient arriver spontanément.

Un exemple simple d'une difficulté inattendue : La balle possède, pour chacun de ses 2 axes, une position, une vitesse ainsi qu'une accélération. L'accélération influe sur la vitesse d'une image à l'autre et la vitesse influe sur la position d'une image à l'autre. Imaginons que la position de la balle se trouve devant un mur et qu'à la frame suivante elle se trouve derrière. A ce moment, on ne peut pas simplement changer le signe de la vitesse, sinon on risquerait de se retrouver avec des images parasites où la balle se trouve derrière un mur. La solution est de repérer de quelle distance la balle dépasse le mur et ensuite de la repositionner à l'intérieur du jeu de cette distance (entre les 2 frames, l'œil fera l'interpolation et même si la balle ne s'est à aucun moment retrouvée "contre" le mur, l'œil sera persuadé du contraire). Le problème ne s'arrête pas là car en la repositionnant la balle, on risque également de dépasser un autre mur. D'où la complexité d'un tel problème. Et là, je ne parle que des murs, les palettes sont encore plus difficiles à gérer. Imaginons que la position de la balle se trouve avant une palette et que la position suivante se trouve après, sans réellement rentrer en contact avec la palette. Selon quelles conditions peut-on savoir si la palette elle était dans la trajectoire ou pas ? Tout en considérant que la palette bouge, elle aussi. J'ai essayé d'analyser ces problèmes de mon mieux et le résultat est plutôt convaincant.

Intelligence artificielle

J'ai profité de ce projet pour m'initier à la création de plusieurs intelligences artificielles basiques, le résultat est plutôt sympa et les niveaux de difficultés assez bien gérés. Le dernier niveau a une IA que l'on pourrait qualifier de "parfaite", il est normal qu'il soit impossible (?) de la battre.

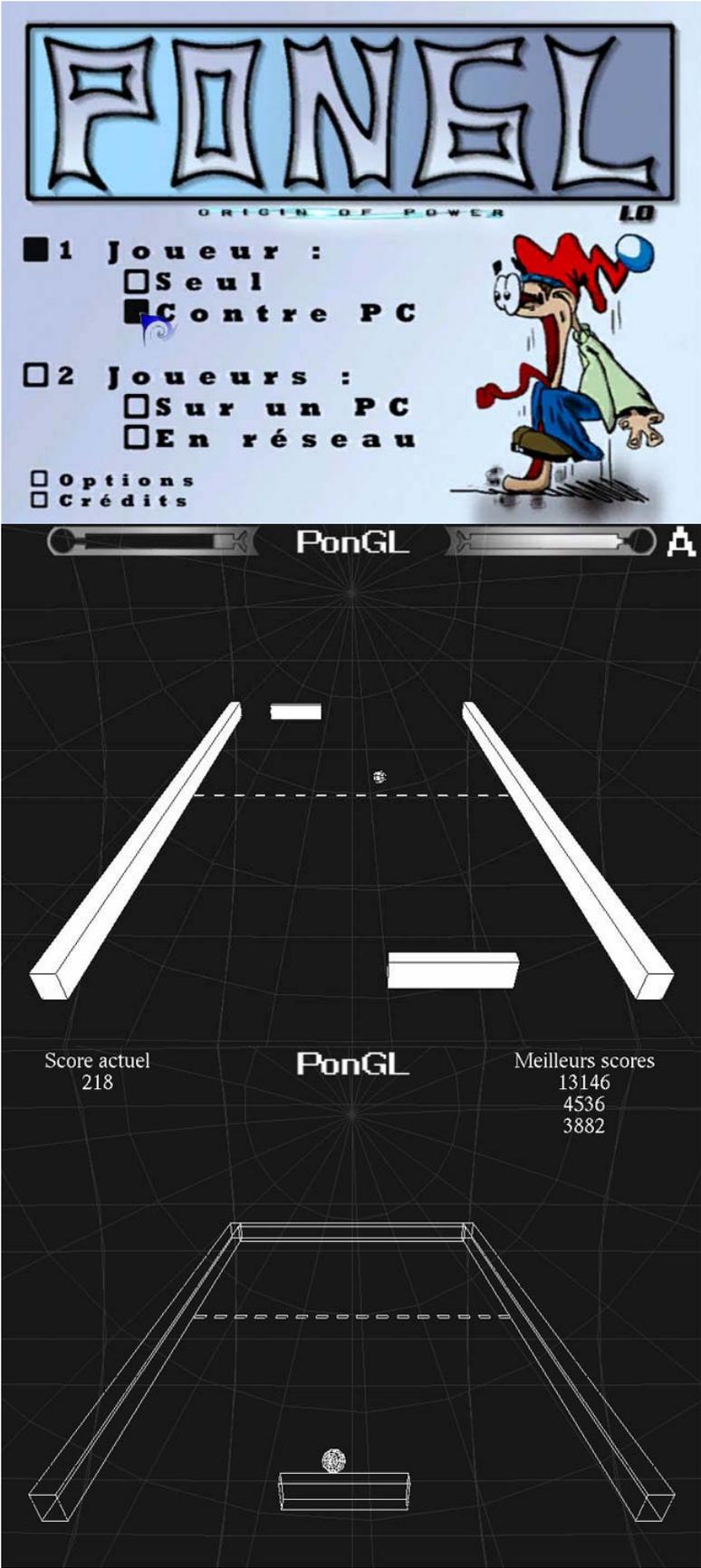
Son

J'ai utilisé une librairie externe pour gérer le son, c'est la librairie SDL_Mixer qui est libre et qui a l'avantage d'être portable. Son utilisation n'était pas très compliquée, le tout est de savoir gérer correctement les événements et d'appeler le bon fichier. Les musiques et bruitages utilisés sont disponibles gratuitement.

Autres

Mis à part les points précédents, il y a encore beaucoup de domaines dans lesquels j'ai dû me débrouiller et improviser un peu. Par exemple pour afficher des textures 2D (les scores, les jauges, ...) au dessus d'un environnement 3D (le jeu en lui-même) ou encore pour gérer le menu des options (mode fil de fer, passage d'une résolution à une autre, activation ou désactivation du son, inertie, accélération de la balle). Il y a pas mal de subtilités pas toujours très simples à résoudre mais qui relèvent plutôt du "bricolage" plutôt que d'une bonne analyse.

Screenshots



ISokoban : Concept

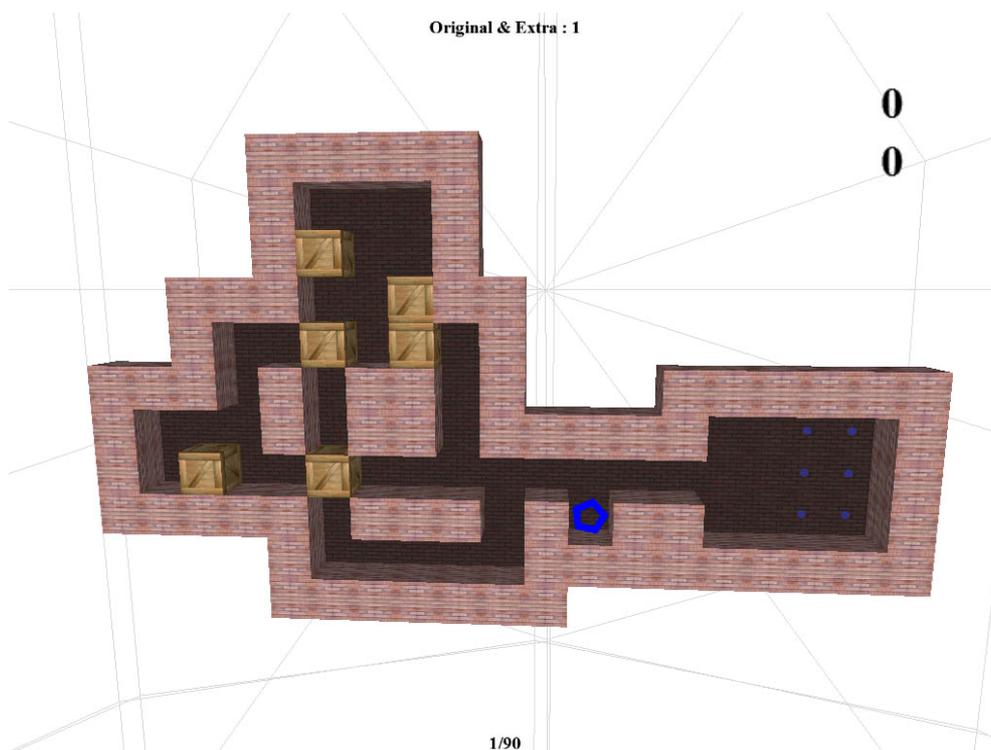
ISokoban est un deuxième jeu que j'ai décidé de faire en essayant de partir d'une base solide. Ainsi, le code est beaucoup plus documenté que celui de PonGL et est mieux pensé.

ISokoban est une adaptation en 3D de Sokoban.

Pour rappel, voici à quoi ressemblait le jeu initial (en 1980) :



Et voici à quoi ressemble ma version :



ISokoban : Fonctionnement

Les règles sont les suivantes (texte de Wikipedia) :

Gardien d'entrepôt (divisé en cases carrées), le joueur doit ranger des caisses sur des cases cibles. Il peut se déplacer dans les quatre directions, et pousser (mais pas tirer) une seule caisse à la fois. Une fois toutes les caisses rangées (c'est parfois un vrai casse-tête), le niveau est réussi et le joueur passe au niveau suivant, plus difficile en général. L'idéal est de réussir avec le moins de coups possibles (déplacements et poussées).

Les touches :

Dans le menu : les flèches droite et gauche servent à passer d'un niveau à l'autre. Plus la pression est longue, plus les niveaux défilent vite. Les touches "enter" et "espace" permettent d'essayer le niveau sélectionné. La touche "ESC" permet de quitter le jeu.

Dans un niveau : Les quatre flèches permettent de déplacer le ISokoban. Si le ISokoban ne bouge pas dans une certaine direction, c'est qu'il y a deux caisses ou un mur qui bloquent. La souris sert, dans une certaine limite, à pivoter la zone de jeu. Les touches "page up" et "page down" sont utiles pour passer au niveau suivant ou au niveau précédent et "home" et "end" servent respectivement à passer au premier ou au dernier niveau. La touche "del" sert à réinitialiser le niveau en cours. "ESC" sert à revenir au menu de sélection des niveaux.

ISokoban : Technique

Timer

Cette fois-ci, j'ai pris mes précautions et j'ai réussi à créer un timer de type "déplacement par temps". Le fonctionnement est simple, avant de tracer chaque frame, on calcule le temps depuis la précédente (en microsecondes) et on utilise la valeur obtenue pour multiplier toutes les valeurs de déplacement. Ainsi, le déplacement va être proportionnel au temps écoulé depuis le dernier affichage. Sur un ordinateur lent, il y aura donc moins d'images, ça sera plus saccadé, mais les positions seront toujours "justes".

Fichiers XML et niveaux ISokoban

Avant de me lancer dans ce projet, je m'étais imposé une condition : rendre le jeu souple au point de pouvoir charger n'importe quel "pack" de niveaux déjà existant. Il n'était pas question de devoir réencoder tous les niveaux un par un à ma manière. J'ai découvert sur Internet qu'il existait un format XML permettant de regrouper un pack de niveaux et diverses infos sur ceux-ci. L'extension de ces fichiers est .slc et il y a moyen d'en trouver pas mal sur internet. Je me suis donc renseigné sur le chargement XML et j'ai trouvé une librairie en C : libxml2.dll. Je l'utilise pour charger les niveaux du jeu dans des structures au lancement de celui-ci. Dans le jeu, le changement d'un pack à un autre est transparent.

Affichage des niveaux

Une fois que le niveau est chargé en mémoire sous un tel format :

```
#####          # -> mur
#   #          $ -> caisse
#$  #          . -> destination
###  $$$      * -> caisse sur une zone de rangement
#   $ $ #      (pas dans ce niveau)
### # ## #    @ -> personnage
#   # ## ##### ..#      + -> personnage sur une zone de rangement
# $ $        ..#
##### ### #@## ..#
#           #####
#####
```

et que chaque "bloc" (caisse, mur, personnage) est créé, il n'est pas trop dur d'afficher les "blocs" sur l'écran les uns après les autres. La seule chose vraiment intéressante est que, pour optimiser l'affichage, j'ai fait en sorte que les faces "cachées" ne soient pas affichées. Par exemple quand une caisse est devant un mur, la face arrière de la caisse et la face avant du mur n'existent pas. On peut gagner ainsi l'affichage de plusieurs dizaines de faces.

Gestion des déplacements

Les déplacements se font dans la mémoire sous forme de caractères ascii (comme le montre le schéma précédent). Le schéma en mémoire évolue grâce à un ensemble de conditions et à chaque étape, un test se lance pour vérifier si toutes les caisses sont sur leurs emplacements. Si c'est le cas, c'est gagné, sinon, on autorise un autre déplacement.

Menus de sélection du niveau

L'air de rien, c'est sûrement la partie la plus complexe du ISokoban. Etant donné que le jeu peut charger n'importe quel "pack" de niveaux, il n'est pas envisageable de stocker sur le disque une texture pour chaque niveau. Au format TGA ça prendrait beaucoup trop de place.

La solution que j'ai trouvée est donc de pouvoir charger "à la volée" une image d'un niveau puis de l'afficher directement à l'écran. D'une manière pratique, chaque fois que le programme aura besoin d'afficher une texture de niveau, il va, dans l'ordre :

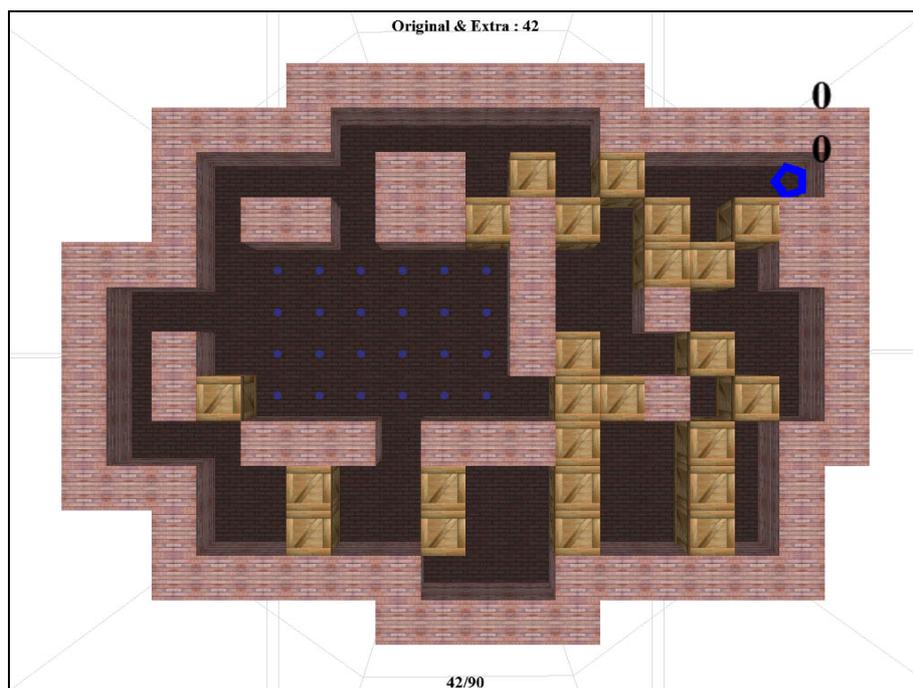
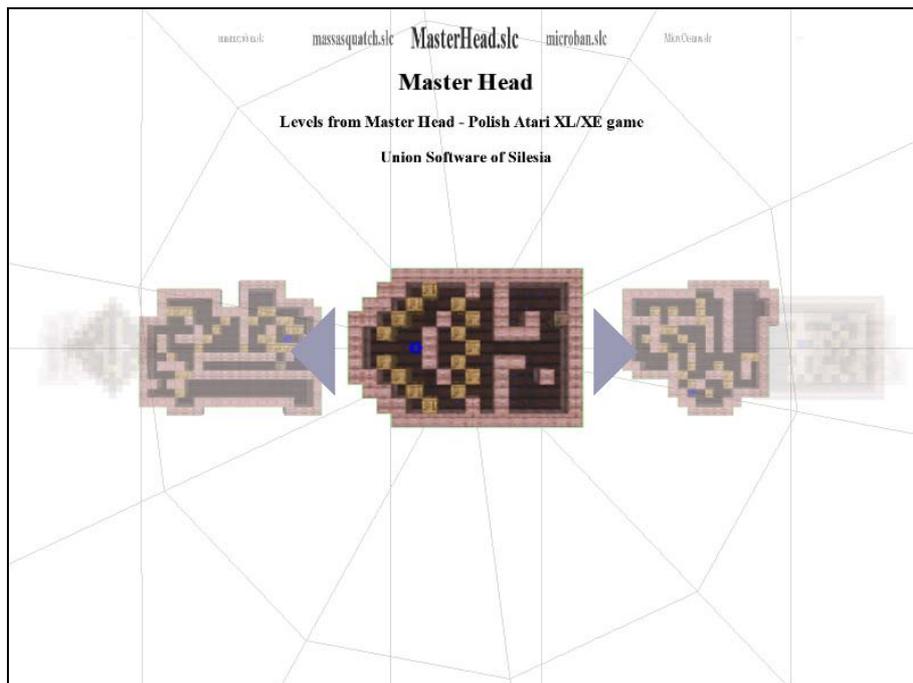
- Charger le niveau en mémoire.
- Afficher (en mémoire) et en basse résolution (128*128) le niveau d'une façon épurée avec un fond vert.
- Capturer l'écran grâce à une fonction OpenGL.
- Transformer tous les pixels verts en pixels transparents.
- Transformer le bloc de données 128*128 pixels en une texture mise en mémoire.
- Effacer l'écran.
- Afficher (à l'écran) le menu en appelant la nouvelle texture.

Toutes ces opérations se font, du moins sur un PC de la puissance du mien, d'une manière totalement transparente pour l'utilisateur. L'avantage de cette méthode est

que le code peut s'adapter pour n'importe quel "pack" de niveaux sans aucune restriction.

Pour le défilement des images, j'ai particulièrement fait attention à la gestion de la mémoire. Chaque fois qu'une flèche est pressée, le niveau suivant est chargé et le niveau précédent est libéré de la mémoire. Ainsi, l'utilisation de la mémoire est toujours identique. Pour gérer les niveaux de transparence d'une manière crédible, je dois forcer l'affichage des images les plus éloignées avant les images les plus proches.

Screenshots



Conclusion

Il est clair que j'adore l'OpenGL et que je me suis investi à fond dans ce domaine. Le projet PonGL était d'ailleurs fini avant même que je termine ma première candidature. Le ISokoban, par contre, a été fait par après et sur de meilleures bases.

C'est plus qu'un simple projet, c'est une véritable passion et j'espère, plus tard, pouvoir trouver un bon compromis entre passion et ambition. Par exemple, en essayant d'approfondir le sujet dans le cadre d'un mémoire de fin d'études.