Rapport du projet Sokoban

Clément Caumes 21501810

09 janvier 2017

1 Description des structures de données

1.1 struct casePlateau ou CASE

La structure **CASE** a deux champs qui correspondent aux différentes possibilités de ce que contient la case. Deux champs étaient nécessaires car il fallait gérer la superposition d'éléments. Le champ *natureCase* peut contenir soit *VIDE*, soit *MUR*, soit *EMPLACEMENT_CAISSE*. Le champ *natureElem* peut contenir soit *PERSO*, soit *CAISSE*, soit *PAS_ELEM*.

1.2 struct positionPerso ou POSITIONPERSO

La structure **POSITIONPERSO** correspond à la position du personnage sur le plateau. Si le personnage n'est pas positionné (lors de l'interface de création) alors les champs x et y contiennent les valeurs *INDEFINI*.

1.3 struct plateau ou PLATEAU

La structure **PLATEAU** possède un champ *CASE** T* qui est un tableau dynamique du plateau à 2 dimensions, un champ *POSITIONPERSO* pour stocker la position du personnage, un champ d'entiers nbCases qui correspond au nombre de cases sur une ligne du plateau. Le nombre de cases réelles du plateau sera donc $nbCases \times nbCases$. Il y a un champ tailleCases qui est la taille des cases dans le plateau.

1.4 struct elem et HISTORIQUE

La structure **HISTORIQUE** contient deux piles. Un élément d'une pile correspond à un *PLATEAU* p, à un entier nbCoups (égal au toucher de caisses) et à un pointeur vers l'élément suivant.

1.5 struct actionCreation ou ACTIONCREATION

La structure **ACTIONCREATION** correspond à l'état du sokoban lorsque l'utilisateur est en train de créer son propre niveau. Il possède un champ d'entiers

mode qui peut être soit *MODE_EDITION* (lors de la création du niveau) soit *MODE_RESOLUTION* (lors du jeu à l'envers) soit *MODE_QUITTER*. Il y a aussi un tableau de caractères nomFichier qui contient le nom du fichier que l'utilisateur a tapé dans la ligne de commande. Enfin, il y a un *PLATEAU* qui sera fixe lors de la création.

1.6 struct actionJeu ou ACTIONJEU

La structure **ACTIONJEU** correspond à l'état du sokoban lorsque l'utilisateur est en train de jouer à une partie d'un fichier. Il possède un champ d'entier mode qui peut être *MODE_JEU* (quand on joue au Sokoban) ou *MODE_QUITTER*. Le nomFichier stocke le nom du fichier sur lequel l'utilisateur est en train de jouer. Le num correspond au numéro du niveau en cours de jeu Le niveauMax est le niveau maximal du fichier. *HISTORIQUE* h permet de stocker l'historique du niveau en cours de jeu.Le nbCases stocke le nombre de cases en longueur (largeur du plateau de sokoban en cours de jeu).

2 Description des algorithmes complexes

2.1 int teste_perso_interieur_entrepot

Ce test renvoie 1 si le personnage est entouré de murs, 0 sinon. Il est composé de 4 sous-fonctions :

- initialise_copie_plateau_test crée un plateau *pCopie* de même taille que le plateau *plat* en argument et met dans chaque case la valeur *CASE_NON_TESTEE*.
- met_valeurs_debut _test met autour de la case en position du personnage de *plat* les valeurs *CASE_TESTEE*.
- diffuse_valeurs_test "transmet" au fur et à mesure la valeur *CASE_TESTEE* dans *pCopie* aux cases voisines si elles ne sont pas des murs (de *plat*).
- test_diffusion_valeurs_test regarde si les cases des bords du plateau de *pCopie* sont *CASE_TESTEE*. Si oui, alors le personnage n'est pas enfermé.

2.2 PLATEAU calcul_plateau_niveau

Cette fonction renvoie un plateau qui correspond à la lecture du niveau num de **ACTIONJEU** a. D'abord, on initialise un plateau de la taille nbCases en argument. Ensuite, il y a une recherche dans le fichier de a.num. Quand on le trouve, on lit à partir de la ligne suivante. Chaque caractère est lu et il y a une incrémentation de x et une décrémentation de y. Mais, à partir du moment où on passe à la ligne dans le fichier, on remet x à 0. Enfin, on retourne le plateau.

3 Améliorations et recherches

— Si on voulait utiliser un minimum de mémoire, il aurait fallu stocker l'historique sous forme de tableau d'éléments. En effet, ce qui change lorsque l'on joue au sokoban sont les caisses et le personnage. Donc, on aurait pu faire ceci : d'abord, **ACTIONJEU** possède un champs suplémentaire *PLATEAU pMur* dans lequel il y a le plateau sans les caisses ni le personnage. Ensuite, lors de l'initialisation de **ACTIONJEU**, on compte le nombre d'éléments *nb* (personnage et caisses) ; on crée un tableau dynamique de la taille *nb*. Puis, on stocke les éléments. Lors des déplacements, on a juste à modifier le contenu des cases du tableau de l'historique.

— Pour être le plus réaliste possible, on pourrait créer une fonction supplémentaire qui teste si il y a des caisses ou des emplacements inacessibles pour le personnage. On pourrait la créer sur le même principe que int teste perso interieur entrepot. Et au lieu de tester si il y a des *CASE_TESTEE* sur les bords, il faudrait regarder chaque case du tableau. Si il existe une case où il y a caisse non testée alors on retourne 0 sinon 1.

— Lors de la création du jeu Sokoban, j'ai essayé de trouver les "failles" de mon programme. En effet, j'ai réfléchi au fait que l'utilisateur peut "nuire" au jeu : en essayant un fichier non valide ou avec des niveaux non valides par exemple. C'est pour cela qu'il faut prévoir toutes les actions possibles de l'utilisateur.