1 Règles du jeu

Source : wikipedia `http://fr.wikipedia.org/wiki/Puissance_4`

"Le but du jeu est d'aligner 4 pions sur une grille comptant 6 rangées et 7 colonnes. Chaque joueur dispose de 21 pions d'une couleur (par convention, en général jaune ou rouge). Tour à tour les deux joueurs placent un pion dans la colonne de leur choix, le pion coulisse alors jusqu'à la position la plus basse possible dans ladite colonne suite à quoi c'est à l'adversaire de jouer. Le vainqueur est le joueur qui réalise le premier un alignement (horizontal, vertical ou diagonal) d'au moins quatre pions de sa couleur. Si alors que toutes les cases de la grille de jeu sont remplies aucun des deux joueurs n'a réalisé un tel alignement, la partie est déclarée nulle."

2 Code source puissance 4

2.1 Ce qu'il faut utiliser

Un code source permettant de représentant le jeu de puissance 4 et d'y jouer est fourni. Il est organisé comme suit :

- case.h : représentation d'une case du jeu (coordonnées de la case dans le jeu, état de la case au cours d'une partie, coup de la partie au cours duquel la case a éventuellement été jouée) ;
- dim.h : représentation des dimensions du jeu (nombre de lignes, nombre de colonnes, taille d'un alignement gagnant) ;
- partie.h : représentation d'une partie du jeu (dimensions du jeu, ensemble de cases, état d'une partie en cours, joueur dont c'est le tour de jouer dans une partie en cours, nombre de coups joués dans une partie en cours).

L'application à développer utilisera donc les 6 fichiers case.h, case.c, dim.h, dim.c, partie.h et partie.c. Néanmoins, **seules les fonctions de l'interface "partie.h" sont amenées à être directement appelées par votre programme** :

- Construction / destruction : `partie_new, partie_free`
- Consultation paramètres du jeu : `partie_get_nbcol, partie_get_nbrow, partie_get_size`
- Consultation partie en cours : `partie_get_etat, partie_get_tourjoueur, partie_get_nbcoup, partie_est_colonne_pleine, partie_get_case_etat`
- Fonctions de jeu : `partie_nouvelle_partie, partie_jouer_colonne, partie_tour_suivant`
- Fonctions annexes : `partie_get_etat_nom, partie_get_statut_nom`

2.2 Guide de lecture

Toutes ces fonctions sont documentées dans le fichier partie.h. Quelques précisions néanmoins :

- Par "pré-condition", on entend condition vérifiée par les paramètres en entrée de fonction et qui n'est donc pas testée par la fonction. Par exemple, pour la fonction `partie_est_colonne_pleine` :
```
/* Pré-conditions :
- partie instanciée (=> dont le tableau partie->plateau)
- 0 <= j < partie->n */
```

```
int partie_est_colonne_pleine(s_partie* partie, int j);
```

Si jamais par exemple `partie == NULL` ou `j == -1`, le programme sort sans aucun doute en erreur (car la fonction tentera des accès du genre `partie->dim` et `partie->plateau[-1]` qui sont invalides). Pour bien utiliser la biliothèque, il faut donc s'assurer de toujours satisfaire les pré-conditions qui sont indiquées.

- Par "post-condition", on entend condition vérifiée par les arguments une fois la fonction terminée, lorsque tout s'est bien passé.

Enfin, quelques précisions sur les fonctions de jeu :

- il n'est pas besoin de faire appel à `partie_nouvelle_partie` avant de lancer la première partie (mais ce n'est évidemment pas génant de le faire) ;
- après avoir joué un coup, si la partie n'est pas finie à l'issue de ce coup, il faut absolument faire appel à la fonction `partie_tour_suivant` (sinon c'est le joueur qui vient de jouer qui aura de nouveau la main au tour suivant) ;
- après avoir joué un coup, si la partie est finie à l'issue de ce coup, il ne faut pas faire appel à la fonction `partie_tour_suivant` (sinon la fonction la fonction `partie_nouvelle_partie` fera commencer l'éventuelle partie suivante par le mauvais joueur).

Pour illustration, un programme permettant de dérouler le jeu en mode console est également proposé : il s'agit du fichier "puissance4_console.c". Pour générer l'exécutable, utiliser le fichier `makefile` fourni :

```
make -f makefile_console
```

3 Travail demandé

Il s'agit de permettre à un utilisateur de jouer au jeu de puissance 4 via une interface graphique. Il faut donc (contrat de base) :

1. D'abord concevoir une interface d'un point de vue purement graphique : quels composants graphiques utiliser, comment les imbriquer entre-eux, quelles sont leurs modalités d'affichage. Ex. pour s'authentifier : il faut une boîte de dialogue, un bouton de validation, un libellé (indiquant "pseudo :") et une zone de saisie pour saisir son pseudo côte à côte, une boite horizontale pour ordonner le libellé et la zone de saisie.
2. Ensuite définir comment cette interface doit interagir avec l'utilisateur : à quel moment du jeu l'utilisateur doit-il ou ne doit-il pas avoir accès à tel ou tel composant de l'interface (que cet accès concerne la capacité de voir ou d'agir). Ex. pour s'authentifier : le bouton ne peut être cliqué que lorsque la zone de saisie comporte au moins 8 caractères.
3. Enfin définir comment cette interface intéragit avec le jeu : quand l'interface doit-elle déclencher une action sur le modèle, comment l'inerface se met-elle à jour suite à une modification du modèle. Ex. pour s'authentifier : lorsque l'utilisateur clique sur le bouton, l'application vérifie que le pesudo saisi fait partie de la liste des peusdo du modèle (imaginons en effet que le modèle comporte un tel annuaire) ; si le pseudo est accepté par le modèle, la fenêtre principale qui aura lancé la boîte de dialogue se personnalise en fonction de ce pseudonyme (mise à jour de la vue suite à l'action de validation demandée au modèle).

Critères d'évaluation (donnés par ordre lexicographique) :

1. La qualité (de la compréhension, du code, de la conception) : mieux vaut en faire moins, mais le faire bien. La qualité du code comprend sa documentation (commentaires) et son organisation (en fonctions, en modules, conçu autour des structures de données qu'il identifie). Elle comprend évidemment aussi sa robustesse technique (compilation sans avertissement, anticipation des comportements de l'utilisateur).

2. La jouabilité de l'interface. On peut en effet aller au-delà du contrat de base, et cela dans différentes directions :

   (a) en tirant partie du modèle proposé (modifier par ex. au cours d'une même exécution du programme les dimensions du jeu) ;

   (b) en enrichissant le modèle (détecter une partie ex-aequo sans que les joueurs ne soient obligés de la dérouler jusqu'au bout, joueur artificiel, ...).

3. L'ergonomie : accès multiple à une même fonctionnalité, accès rapide, *etc.*, permettant de satisfaire la plus grande variété possible de profils d'utilisateurs.

4. L'inventivité : graphisme, scénario de mise en scènes, enrichissements originaux de l'interface (*ex.* parties rapides), ...

4 Règles du jeu... du projet

4.1 Groupes

Deux personnes maximum par groupe. **Attention : projets similaires sur n groupes $\Rightarrow$ note divisée par n.**

4.2 Échéances

- Livraison le mercredi 18 mai avant minuit
- Soutenances la semaine du lundi 23 mai (présence **obligatoire**).

4.3 Contenu de la livraison

- Le code source.
- La documentation : recueil des analyses qui ont mené à vos différents choix graphiques (composants) et techniques (structures, gestion événementielle).
- Aide à l'installation (un fichier makfile peut suffire).

4.4 Mode de livraison : électronique

- Par mail, au responsable de cours et à votre chargé de TD/TP, en prenant bien soin de spécifier, dans le sujet, "IIG - projet" et dans le corps du message, les noms des membres du groupe + Groupe de TD d'option.
- Dans l'idéal (mais ce n'est pas obligatoire), transmettre un lien de téléchargement plutôt qu'une pièce jointe.