Michael SCHUH
AS 2008 – 2009

MINI-PROJET COLUMNS

I) Objectifs

Le projet consiste en une réécriture en Borland Pascal du jeu Columns paru en 1990.
Dans ce jeu le joueur doit créer, à l'aide de piles de trois joyaux tombant du ciel, des alignements d'au moins trois joyaux de même couleur pour les faire disparaître.
Une approche des règles du jeu original est visible dans le pdf joint au dossier.

Le programme présenté est écrit en Borland Pascal. Les éléments graphiques sont des BitMap 256 couleurs tirés de captures d'écrans du jeu original émulé avec un émulateur de jeu. Les éléments sonores sont également tirés du jeu original.

– Objectifs atteints :

- Le joueur peut piloter les joyaux tombant du ciel et créer des alignements qui disparaissent.
- Les éléments de décor du jeu original (décor, joyaux) ont été partiellement implémentés, ainsi que certaines animations (explosions) et quelques sons.
- Calcul du score, nombre de joyaux éliminés de l'aire, niveau à la difficulté croissante.
- Animations : outre les explosions, l'aire de jeu est partiellement animée, et les blocs de joyaux restant retombent case après case une fois qu'un joyau a explosé néanmoins...

– A faire :

- … Dans le jeu original, les joyaux tombent demi-case par demi-case, et non case par case. Cela reste à implémenter.
- Tableau des meilleurs scores
- Niveaux de difficultés
- Implémenter le joyau magique (cf présentation du projet, décembre 2008)
- etc...

II) Ecriture du programme

Bref descriptif des unités :

UCellule : unité de base, définit la classe CCellule. La CCellule a pour attribut ses coordonnées (ligne, colonne) dans une matrice, ainsi qu'un contenu (couleur ou vide (Nul)). Ses méthode lui permettent de transférer du contenu d'une cellule à une autre par exemple.

UMatrice : définit la CMatrice, tableau à deux dimensions de 50x50 pCCellule. Toutes ne sont pas utilisées (elles ont pour valeur nil) ce qui nous permet de créer des tableaux à deux dimensions de dimensions choisies (ici c'est 13x6).

UJoyau : définit le CJoyau , une classe d'objet qui se déplace dans la matrice. Ses méthodes lui permettent d'être mobile. Son contenu est écrit sur la marice.

UPile : définit la CPile, une classe d'objet qui consiste à empiler 3 joyaux solidaires verticalement pour qu'ils se déplacent dans une matrice.

UTest : UTest contient des algorithmes parcourant la matrice case après case en comparant chaque case à sa case voisine à l'aide d'une procédure récursive afin de dénicher des alignements d'au moins trois joyaux de même couleur (cf III,1)

UDMatVid : dessin de matrice vide.

UDJoy : dessin de joyaux

UDAire : dessine la matrice avec ses joyaux, en fonction de la position des joyaux sur la matrice

UDPile : dessin d'une pile en mouvement.

UApercu : l'aperçu de la prochaine pile pendant la partie.

UDAireAn : animations dans l'aire de jeu, soit les explosions et les effondrements.

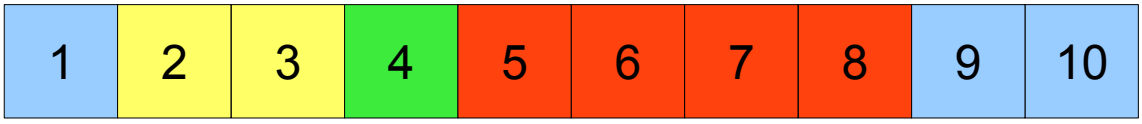
UJeu3 : déroulement du jeu.

III) Quelques spécifications

III.1) Recherches d'alignements

Supposons des cases de couleurs différentes caractérisées par leur numéro et rangées par ordre chronologiques (1, 2, 3, 4, …).
On cherche les alignements de trois couleurs ou plus sur cette série de cases voisines.
Pour ce faire, on compare la case 1 à la case 2, et si les cases 1 et 2 ont la même couleur, alors on incrémente un compteur précédemment réinitialisé et on compare 2 et 3, s'ils ont la même couleur, on incrémente encore le compteur, et ainsi de suite.
Le mécanisme est récursif et permet de compter des séries identiques.

Illustrons cette explication avec l'exemple suivant :



On souhaite trouver sur cette rangée des alignements d'au moins trois cases de même couleur.
On va utiliser l'algorithme Compare (i, i + 1) où i et i + 1 sont des numéros de cases, et la variable n qui va compter les séries de même couleur
Soit :

```
Compare (i, i + 1)
DEBUT
        SI Couleur (i) = Couleur (i +1 ) ALORS
                n ←  n + 1
                Compare (i + 1, i + 2)
                SI n >=3 ALORS Marquer (i) FIN SI
        FIN SI
FIN
```

Cet algorithme est récursif : si un premier élément a sa couleur identique avec un second, alors la fonction va s'appeler elle-même et la couleur du second va être comparée à celle du troisième élément.
Comme on veut trouver tous les éléments de la rangée, on va mettre Compare dans une boucle avec une variable qui va parcourir toute la rangée :

```
a ← 0
TANT QUE a < 10 FAIRE
        n ← 1
        a ← a + n
        Compare (a, a + 1)
FIN TANT QUE
```

On va effectuer une trace de l'algorithme :

1) On part de a = 1, on compare 1 et 2, ils ne sont pas pareils (n = 1)
2) On compare a = 2 et 3, ce sont les mêmes, on compare 3 et 4, ils sont différents (n = 2)
3) On compare a = 2 + 2 =4 à 5 et ils sont différents (n = 1)
4) On compare 5 à 6, ce sont les mêmes, 6 et 7 sont les mêmes, 7 et 8 sont les mêmes, 8 et 9 ne sont pas les mêmes. 5, 6, 7, 8 sont marqués. (n = 4)
5) a = 5 + 4 = 9 et 10 ne sont pas les mêmes

Cet algorithme nous a permis de trouver un alignement de 4 cases de même couleur. C'est en gros cette méthode qui est utilisée dans UTest (voir la procédure Vertical).

III.2) Effondrements

Dans le jeu Columns, lorsque des joyaux disparaissent, l'espace laissé vaquant est occupé par les joyaux du dessus.

Fig 1

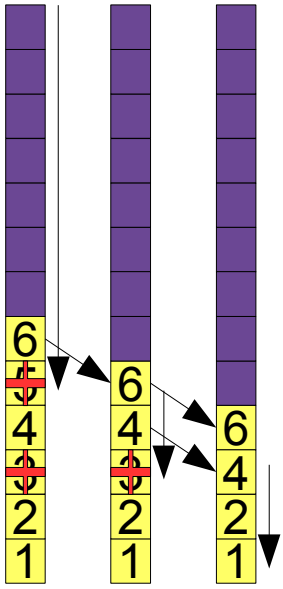


Fig 2

Fig 1 : On va ici considérer la 2e colonne en partant de la gauche. Les 3e et 5e joyaux en partant du bas disparaissent.
Fig 2 : Nous schématisons ici ce qui se passe avec la deuxième colonne considérée dans la figure 1. Les joyaux 3 et 5 disparaissent, on les marque avec une croix rouge (à gauche).
Une variable compteur (boucle while) parcourt la colonne tant qu'elle ne repère pas de joyaux qui disparaisse.
Quand elle en repère un (le 5e), elle le supprime (valeur=Nul) et tous les joyaux au dessus descendent d'une ligne (milieu).
On continue alors de parcourir la colonne jusqu'à ce qu'on trouve un autre joyau disparaissant (le 3). Il est alors retiré et tous les joyaux qui se trouvaient au dessus du 3 descendent d'une ligne (on aboutit au schémas de droite)

Lors d'un effondrement est créé des « piles tombantes » pour les besoins des animations.

III.3) Les « piles tombantes »

On assigne dans la Matrice la valeur Nul aux cellules qui disparaissent.
Puis, en parcourant la colonne de bas en haut on repère les cellules vides.
Une fois la cellule vide repérée, on continue le parcourt jusqu'à ce qu'on trouve une cellule occupée (au besoin on a recourt à une fonction récursive pour compter les cellules vides).
Dès qu'on trouve une cellule occupée, on continue le parcourt de la colonne jusqu'à trouver une cellule vide. Avec le contenu des cellules occupées va pouvoir être créé une pile qui s'animera et tombera selon un certain laps de temps. Ces piles sont construites avec le constructor CDPile.Tombant, mais c'est dans UDAireAn que sont écrites les procédures d'animation. Dans l'illustration de la colonne ci-dessus, les joyau 4 et le joyau 6 formeraient chacun une pile tombante, qui tomberaient dans les interstices laissés vacant.

III.4) Les « piles restantes »

Lors d'une partie de Columns, il arrive souvent que le joueur débordé parvienne à créer un alignement avec l'un des joyaux de la pile (celui de la base ou le deuxième). Le reste de la pile resté en dehors va alors former une « pile restante » qui va être initialisé en même temps que les piles tombantes pour les besoins de l'animation et de l'esprit de la règle originale.