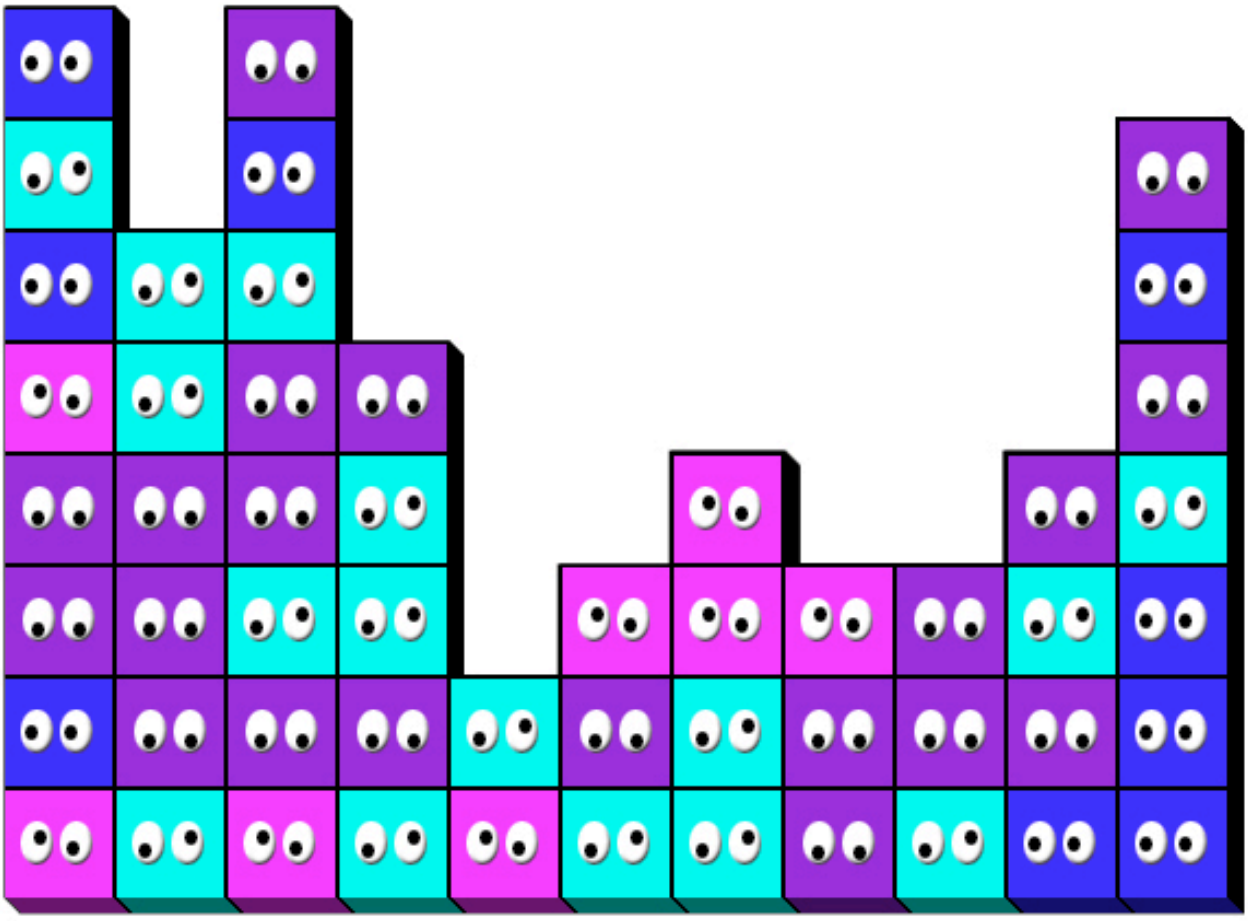
MARECHAL ANTHONY
MARECHAL BENOIT

THE GAME 2K4



PROJET I2A-I2B 2004

SOMMAIRE

1ère partie : cahier des charges

1. Sujet

Le jeu à réaliser se déroule dans le cadre « vertical » rempli de pièces carrées chacune dotée d'un motif.

Au début de chaque partie, les pièces sont tirées aléatoirement dans un ensemble prédéfini et réduites au nombre de motifs choisi par l'utilisateur (ou par le mode dans lequel il se trouve « Facile » 3 motifs, « Moyen » 4 motifs, « Difficile » 6 motifs). De ce fait, le terrain de jeu d'une nouvelle partie comprend en général des zones uniformes définies comme des zones constituées de pièces identiques contiguës.

A chaque instant du jeu (coup), le joueur peut sélectionner et détruire une zone en cliquant sur une des pièces qu'elle contient à condition qu'elle comprenne au moins deux pièces. La destruction d'une zone provoque l'écroulement des pièces situées au dessus ou le regroupement vers la gauche des pièces situées à droite de la zone si la destruction de cette dernière provoque la disparition d'une ou plusieurs colonnes.

L'objectif du joueur est de laisser à la fin le moins possible de pièces isolées sur le terrain de jeu, en sélectionnant judicieusement les zones successives. A la fin du jeu, la machine affiche le score du joueur et lui propose de jouer une nouvelle partie ou de la recommencer.

2. Objectifs

Le programme doit :

- être réalisé en JAVA SWING.
- scruter si le jeu est fini, c'est à dire si plus aucun coup n'est possible.
- afficher le score définie en fonction :
  - de la taille des blocs supprimés.
  - des pièces restantes.
  - du nombre de coups joués.
- proposer des options de jeu telles que :
  - le choix de la dimension du jeu.
  - le choix du nombre de pièces et de leurs motifs.

3. Objectifs supplémentaires

Le programme gère :

- La possibilité de revenir sur autant de coups joués que l'on veut.
- Le choix de 3 modes de difficultés (Facile, Moyen, Difficile).
- L'enregistrement des Meilleurs Scores pour chaque mode de difficultés.
- L'affichage de la fenêtre de jeu de manière toujours optimisée :
  - Centrage et redimensionnement automatique de la fenêtre par rapport à l'écran.

- Choix automatique de la taille des motifs (3 tailles différentes) selon la taille de la grille choisie.
- Le mode Smiley's dans lequel les règles ont été redéfinies :
  - nombre de pièces différentes : 4.
  - motifs : Smiley's.
  - grille :12 x 16.
  - nombre de coups possibles 65.
  - compte à rebours : 40 secondes.
  - objectif unique : ne laisser aucune pièce en fin de jeu.

2ème partie : modélisation

1. Analyse

Pour gérer le jeu nous avons choisi de distinguer deux classes :

- la classe **Cjeu** permettant la gestion de la grille de jeu.
- la classe **Cinterface** permettant quant à elle la gestion de l'interface du jeu.

A. La classe Cjeu

La classe **Cjeu** est en fait le cœur du programme. Elle possède un tableau d'entiers représentant la grille de jeu avec des nombres compris entre 0 et le nombre de motifs choisis par le joueur. Les 0 représentent les cases vides (ayant été supprimées) et les nombres représentent chacun un motif qui leur est propre.

C'est dans cette classe et sur ce tableau que nous avons pu définir différentes méthodes permettant de :

- générer aléatoirement chaque nouvelle partie.
- calculer les coordonnées d'un bloc sélectionné.
- gérer leur suppression puis la « descente » (ou la « translation » vers la gauche) des pièces supérieures (ou à droite) du bloc supprimé.
- compter le nombre de coups joués.
- créer la liste des tableaux pour la fonction permettant de revenir sur un ou plusieurs coups déjà joués.
- calculer le score réalisé pour chaque bloc supprimé.

Nous reviendrons plus longuement sur cette classe et sur les différentes méthodes qu'elle possède dans la partie Programmation.

B. La classe Cinterface

La classe **Cinterface** gère la totalité de l'interface graphique du jeu. Elle régit aussi bien la fenêtre principale que les fenêtres annexes (ex : affichage des scores, choix des motifs, choix de la taille de la grille, etc.).

Cette classe possède un attribut de type **Cjeu** lui permettant d'interagir avec cette dernière. Ainsi, cette classe peut « traduire » les informations de la classe **Cjeu** et les interpréter graphiquement grâce à différentes méthodes telles que :

- La création de la grille de jeu (en fonction du tableau d'aléatoire créé dans **Cjeu**).
- L'actualisation de la grille de jeu (après avoir cliqué sur un bloc valide) .
- La gestion de la fin du jeu (avec calcul du score final et enregistrement dans les Meilleurs Scores si le record a été battu).

Cette classe dispose également d'autres méthodes pour la sauvegarde des données, et deux autres classes héritières de la classe Thread permettant le défilement du score final et le

compte à rebours du mode Smiley's. Ces deux classes étant hors programme nous ne les détaillerons pas plus ici.

Nous reviendrons plus longuement sur la classe **Cinterface** dans la partie Programmation et dans la troisième partie consacrée à l'interface graphique.

2. Programmation

Nous allons faire dans cette partie une description plus précise des classes avec les attributs et les méthodes. Aussi, pour les méthodes complexes nous donnerons une explication détaillée de leur fonctionnement ainsi que leur code commenté en Java.

1. La classe Cjeu en détaille.

A. Les attributs

La classe **Cjeu** possède plusieurs attributs qui serons exploités par la suite par la classe **Cinterface**. Ils permettent le suivi du jeu au cours de ses différents états, et donnent plusieurs informations sur ce dernier. Vous trouverez une liste exhaustive de tous les attributs de la classe **Cjeu** ci-après :

- Un tableau d'entiers à deux dimensions nommé ***jeu***.
- Un tableau stockant les coordonnées de toutes les cases d'un bloc nommé ***tmp***.
- Deux entiers ***nbL*** et ***nbC*** correspondant respectivement au nombre de lignes et de colonnes du tableau ***jeu***.
- Un entier ***nbMotif*** correspondant au nombre de motifs différents à créer dans le tableau ***jeu***.
- Un entier ***nbcoup*** correspondant au nombre de coups déjà effectués par le joueur.
- Un entier ***taille_bloc*** correspondant au nombre de pièces qui constituent le bloc à supprimer.
- Un entier ***taille_tmp*** correspondant au nombre de colonnes du tableau ***tmp***. Cet attribut est défini en fonction du nombre de lignes et de colonnes qui constituent la grille de jeu par l'équation suivante :

  taille_tmp = (nbL * nbC) / 2 ;

  Ainsi, la taille du tableau stockant les coordonnées d'un bloc est toujours optimisée en fonction des grilles de jeux.
- Une chaîne de caractères ***inf*** correspondant au différentes informations données au joueur selon l'endroit où il a cliqué.

  (ex : « Bloc de 1: Non supprimable! » ou bien encore « C'est une zone vide. »)
- Un entier ***score_bloc*** défini par la méthode **calcul_score()** par l'algorithme suivant :

```
score_bloc=(taille_bloc*25)+((taille_bloc*25*(taille_bloc-2)*10)/100);
if (taille_bloc>7) score_bloc+=50; //Super-Bloc (+50pts)
if (taille_bloc>13) score_bloc+=100; //Giga-Bloc (+150pts)
if (taille_bloc>19) score_bloc+=200; //Ultra-Bloc (+350pts);
```

Cet algorithme permet de donner un nombre de points avec une majoration de plus en plus importante selon la taille du bloc. Il permet également de donner des bonus supplémentaires si une certaine taille de bloc est dépassée (ex : un bloc de plus de 19 pièces apporte un Bonus de 350 points !).

- Un entier ***score_bloc_total*** correspondant à la somme de tous les ***score_bloc*** déjà réalisés dans la partie.
- Un entier ***nbbloc21*** (nombre de bloc de un) correspondant au nombre de pièces restantes en fin de partie. Cet attribut est défini par la méthode **nb_bloc_de_1()** qui compte en fin de jeu toutes les cases du tableau ***jeu*** différentes de 0 (donc correspondant au nombre de pièces restantes).
- Une *ArrayList* ***v*** stockant tous les tableaux déjà joués au cours de la partie. Ce qui permet de revenir sur un ou plusieurs coups joués.
- Une *ArrayList* ***score*** stockant la valeur du score à chaque coup joué (liste indispensable lors de la restauration d'un coup pour éviter toute triche).

B. Les méthodes

Nous allons maintenant présenter les différentes méthodes que comprend la classe **Cjeu**. On commencera tout d'abord par expliquer la création du tableau ***jeu***, nous développerons ensuite la fonction relative à un clic sur une pièce, puis nous verrons en détail le fonctionnement du calcul des coordonnées des pièces d'un même bloc, enfin nous apporterons des précisions sur les méthodes gérant les coups précédents et la fin du jeu.

➢ Initialisation de la classe Cjeu et création du tableau *jeu*

- Le constructeur

La classe **Cjeu** possède un constructeur avec 3 paramètres :

- ***nblignes*** correspondant au nombre de lignes du futur tableau ***jeu***.
- ***nbcolones*** correspondant au nombre de colonnes du futur tableau ***jeu***.
- ***nbmotif*** correspondant au nombre de valeurs différentes à mettre dans le tableau ***jeu***.

Le constructeur affecte alors ces valeurs à ***nbL***, ***nbC***, ***nbMotif*** respectivement. Ensuite il affecte la valeur à ***taille_tmp*** (grâce à l'équation expliquée plus haut), puis il initialise le tableau ***tmp*** de 0 et alloue le tableau ***jeu*** au nombre de lignes et de colonnes définies. Enfin il lance la méthode **initialise_jeu()** .

- La méthode **initialise_jeu()**

Cette méthode remplit le tableau ***jeu*** de nombres aléatoires compris entre un et ***nbMotif***. Puis elle crée une bordure de 0 tout autour du tableau (NB : en effet les attributs ***nbL*** et ***nbC*** ont été en réalité initialisés à nbL=nblignes + 2 et nbC=nbcolones+2 nous verrons pourquoi dans l'explication de la méthode **calcul_bloc()**).

➢ Méthode relatif à un clic sur une pièce

A chaque clic sur une pièce (correspondant à un bouton dans l'interface) est lancée la méthode **action().** Cette méthode fait appel à plusieurs fonctions permettant le stockage des coordonnées des pièces d'un même bloc, la « descente » ou la « translation vers la gauche » des pièces supérieures ou à droite du bloc sélectionné, le calcul du score du bloc, et enfin le stockage (dans l'*ArrayList **v***) du tableau tel qu'il était avant la suppression du bloc ainsi que le score (dans l'*ArrayList **score***).

Vous pouvez voir son code commenté en java ci-après :

```
public void action (int l,int c) { // 'l' et 'c' sont la ligne jouez  ET la  colonne joué
        taille_bloc=0;  // initialisation de taille_bloc à 0
        for (int i=0; i<2 ;i++) for(int j=0 ; j<taille_tmp ; j++)
                { tmp[i][j]=0; } //initialisation à 0 du tableau tmp stockant les coordonnées du bloc.
        l++;   // coordonnée du bouton +1 (car mon tableau jeu est en réalité plus grand )
        c++;   // idem
        tmp[0][0]=l;  // Saisie des coordonnées du bouton cliquez
        tmp[1][0]=c;  // dans la 1iere colonne du tableau tmp
        val=jeu[l][c]; // Récupération de la valeur de la case sélectionnée
        if (val !=0) { //si la case sélectionnée n'est pas vide
                taille_bloc++ ;
                calcul_bloc(l,c);  // Lancement de la fonction récursive pour calculer le bloc
                if (taille_bloc >1) { //si le bloc est >1 (supprimable)
                Precedent(); // enregistrement du tableau dans l'ArrayList v
                nbcoup++;    // incrémentation du nombre de coup joué
                score_bloc = calcul_score() ; //calcul du score
                score_bloc_total+=score_bloc;
                remplace_bloc(); //remplace les pièces du bloc par des 0
                interverti();    // permet de faire descendre les pièces situées au dessus du bloc*
        for(int k=1;k<nbC-1;k++)
        for(int i=1;i<nbC-1;i++) if(jeu[nbL-2][i]==0)
        decalage_Horizontale(i); //pour l'éventuel décalage horizontal (vers la gauche)
                /* informations selon la taille du bloc sélectionné*/
        if (taille_bloc<8)
                inf = "Bloc de "+taille_bloc;
                 else if (taille_bloc>7 && taille_bloc<14)
                        inf = "Bloc de "+taille_bloc+" ==> Super-Bloc !" ;
                 else if (taille_bloc>13 && taille_bloc<20)
                        inf = "Bloc de "+taille_bloc+" ==> GiGa-Bloc !!";
                 else if (taille_bloc>19)
                        inf = "Bloc de "+taille_bloc+" ==> ULTRA-Bloc !!!";
                    }
                 else inf = "Bloc de 1: Non supprimable!" ;
                   }
                 else inf = "C'est une zone vide";
            }
```

Quelques précisions sur le fonctionnement des fonctions **remplace_bloc()** et **interverti( ) *:***

- Pour faire « descendre » les pièces situées au dessus d'un bloc qui doit disparaître on utilise deux fonctions.
- La fonction **remplace_bloc()** remplace les cases du bloc (dont les coordonnées ont été stockées dans le tableau ***tmp***) par des 0.

- Puis la fonction **interverti()** parcourt le tableau ***jeu*** ainsi modifié puis dès qu'elle trouve la valeur 0 elle l'intervertit avec la case située au dessus.
- Ainsi, après plusieurs exécutions de cette fonction la « descente » des pièces situées au dessus est réalisée.

➢ Stockage des coordonnées des cases appartenant à un même motif (calcul du bloc)

Pour la création du tableau (tableau ***tmp***) stockant les coordonnées des pièces appartenant à un même motif nous avons choisi d'utiliser un appel récursif d'une fonction nommée **calcul_bloc()**. Cette dernière s'exécute tant qu'elle trouve des nouvelles pièces de mêmes motifs que celui cliqué au départ.

Pour son bon fonctionnement, elle fait appel à deux fonctions :

- **verif()** qui vérifie si les coordonnées de la case de même motif n'ont pas déjà été rentrées dans le tableau ***tmp***. (cette fonction risquait de faire des erreurs *OutOfBoundsException* dans le tableau jeu d'où l'allocation de lignes et de colonnes supplémentaires pour le tableau jeu - rappel : nbL = nblignes + 2 et nbC = nbcolone+2).
- **rempli_tmp()** qui entre les coordonnées d'une case dans le tableau ***tmp***.

Vous trouverez ci-après le code commenté en Java de cette fonction :

```
public void calcul_bloc (int l, int c) { // "l" et "c" sont les coordonnées de la case

  if (taille_bloc < taille_tmp) { // on vérifie que le tableau tmp est assez grand
                                                    // pour contenir de nouvelles coordonnées
  int i,j;
  if (jeu[l][c] != 0 ) { // si la case n'est pas une zone vide
                  if  ( (jeu[l-1][c]==val) && (verif(l-1,c)==true) ) {
                // si la case située au dessus de cette case et est de même valeur
                // et si cette case n'as pas déjà été entrée dans le tableau tmp
                        rempli_tmp(l-1,c); // on entre ces coordonnées dans tmp
                        calcul_bloc(l-1,c); // on relance la fonction sur cette case

                  }

        // on refait ces vérifications pour les 3 autres cases autour de cette case
                  if ( (jeu[l][c+1]==val) && (verif(l,c+1)==true) ) {
                        rempli_tmp(l,c+1);
                        calcul_bloc(l,c+1);
            }

                  if  ( (jeu[l+1][c]==val) && (verif(l+1,c)==true) ) {
                        rempli_tmp(l+1,c);
                        calcul_bloc(l+1,c);
                  }
                  if ( (jeu[l][c-1]==val) && (verif(l,c-1)==true) ) {
                        rempli_tmp(l,c-1);
                        calcul_bloc(l,c-1);
                  }
  }
```

➢ Méthodes permettant la gestion des coups précédents et la fin du jeu

- Méthodes gérant les coups précédents

A chaque coup entraînant la suppression d'un bloc le tableau est sauvegardé dans une *ArrayList*. Pour ce faire, on utilise une fonction nommée **Precedent()** dans laquelle on recopie le tableau ***jeu*** en cour dans un tableau temporaire que l'on stocke ensuite dans l'*ArrayList* ***v*** . Ainsi il suffit de spécifier quel indice de l'*ArrayList* on veut pour avoir accès au tableau souhaité et de le recopier dans le tableau ***jeu***.

Une méthode **vide_list()** nous sert à chaque nouvelle partie pour vider la liste de tableaux.

- Méthodes scrutant la fin du jeu

A chaque coup joué une fonction **game_over()** est lancée. Cette fonction renvoie un booléen indiquant si la partie est fini ou non.

Pour le savoir elle parcourt toutes les cases du tableau ***jeu*** en regardant si la case située à sa droite et la case située en dessous d'elle est de même valeur qu'elle, si c'est le cas elle renvoie « *false* » car cela signifie que le jeu n'est pas encore fini, sinon elle renvoie « *true* ».

2. *La classe Cinterface en détail*

A. Les attributs

La classe **Cinterface** possède un attribut de type **Cjeu** nommé ***TheGame*** qui lui permet d'interagir avec la classe **Cjeu** pour la gestion de la grille de jeu. Vous trouverez ci-dessous une liste exhaustive de tous les attributs de cette classe.

La classe **Cinterface** possède :

- Un attribut de type **Cjeu** appelé ***TheGame***.
- Deux entiers ***nbL*** et ***nbC*** pour le nombre de lignes et de colonnes (ces deux attributs sont initialisés à 6 lors du lancement du jeu).
- Un tableau d'entiers ***jeu*** similaire à celui de la classe **Cjeu** (le tableau jeu de la classe **Cjeu** est privé).
- Un entier ***nbMotif*** indiquant le nombre de motifs différents pour la grille de jeu (initialisé à 3 lors du lancement du jeu).
- Un entier ***taille_motif*** correspondant à la taille des motifs. Nous avons en effet créé tous les motifs dans trois tailles différentes pour que la jouabilité reste toujours bonne même avec des grandes grilles (les tailles sont : 50x50 pixels, 40x40 pixels et 30x30 pixels). La gestion de la taille des motifs se fait automatiquement, cependant le joueur peut changer leur taille s'il le souhaite lors d'une partie personnalisée.
- Une chaîne de caractères ***formeMotif*** correspondant à la série de motifs choisis (il en existe trois pour les modes normaux et une pour le mode Smiley's).
- Un booléen ***fini*** qui indique si la partie est terminée ou non.

- Un entier ***coups_pre*** qui indique le nombre de coups précédents autorisés (initialisé à 1000 par défaut).
- Un entier ***cpt_pre*** qui compte le nombre de coups précédents effectués. Il est en effet possible de spécifier le nombre de coups précédents tolérés par le joueur.
- Un entier ***tt*** correspondant au score final réalisé par le joueur. Ce score tient compte du nombre de coups effectués par le joueur et du nombre de pièces restantes.
- Un attribut de type File nommé ***fichier*** qui indique le nom du fichier à créer pour les Meilleurs Scores. Ce fichier est appelé *HightScore*.
- Un attribut de type Chrono nommé compte_a_rebour. Cet attributs est utile pour le mode Smiley's dans lequel le joueur possède seulement 40 secondes pour gagner.
- Un attribut de type defil_score nommé ***defil*** qui permet le défilement du score final lors du lancement de la fenêtre du calcul des scores. Le score part de zéro et augmente rapidement jusqu'au score réalisé par le joueur.

NB : Pour plus de souplesse et pour ne pas avoir à déclarer à nouveau certains attributs de la classe **Cjeu** nous avons décidé de les déclarer public.

B. Les méthodes

Voyons à présent les différentes méthodes qui constituent la classe **Cinterface**. Dans un premier temps nous développerons les méthodes relatives à l'initialisation et à la construction du jeu, puis nous verrons de manière détaillée comment la grille de jeu s'actualise à chaque suppression de bloc, enfin nous verrons les fonctions utiles à l'enregistrement des meilleurs scores et la gestion de la fin d'une partie.

➢ *Initialisation de la classe Cinterface et création de la grille de jeu.*

✓ Le constructeur

Le constructeur de cette classe est très simple. Il ne prend aucun paramètre et se contentent de :

- lancer la fonction **initComponents().** Pour la création de l'interface (fonction créé par l' IDE).
- lancer la création du fichier *HightScore*, par la fonction **creation_fichier()**.
- implémenter la variable ***TheGame*** de type **Cjeu** avec les paramètres ***nbL***, ***nbC*** et ***nbMotif***.
- lancer la fonction **creation_jeu().**

✓ La fonction creation_jeu()

Dans un premier temps la fonction **creation_jeu()** initialise les différentes variables relatives à une nouvelle partie. Puis elle vérifie le nombre de colonnes et de lignes qui vont être créés, lui permettant de redimensionner et de centrer la fenêtre principale de manière optimisée. Enfin elle crée les boutons de la grille de jeu en donnant à chacun un nom et un écouteur.

Voici le code commenté en Java de la génération des boutons :

```
« for(int i = 0 ; i<nbL ; i++) // parcours de tout le tableau
for(int j=0 ;j <nbC ; j++) {
  JButton button = new JButton() ;
                    // on donne un numéro à chaque bouton
                    button.setName(String.valueOf((i*nbC)+j));

                    // on crée l'écouteur
  button.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt){ buttonActionPerformed(evt);}});

  // on ajoute le bouton à la grille
   jPanelJeuGrille.add(button);
            } »
```

➢ *Actualisation de la grille de jeu après suppression d'un bloc*

Lorsque le joueur clique sur une pièce (un bouton) il lance la méthode action de **Cjeu** expliqué plus haut. Cette fonction modifie alors le tableau ***jeu*** en conséquence. Pour actualiser la grille de jeu par le nouveau tableau ainsi modifié, nous utilisons la fonction **copi_tableau_Cjeu()**.

Dans un premier temps la fonction actualise l'interface (pour le nombre de coups et le score), dans un deuxième temps elle lance l'apparition de gifs animés en fonction de la taille des blocs, et enfin elle actualise chaque bouton par son nouveau motif en fonction du tableau ***jeu***.

Voici le code commenté en Java de la fonction **copi_tableau_Cjeu()** :

```
public void copi_tableau_Cjeu() {

// initialisation de l'interface
   AfficheScore.setText(String.valueOf(TheGame.score_bloc));
   AfficheCoups.setText(String.valueOf(TheGame.nbcoup));
// récupération des informations selon l'endroit cliquez (« bloc de 1 », //« zone vide »)
   infos.setText(TheGame.inf);

// Effet esthétique : des gifs sont lancés en fonction de la taille des blocs cliqués

   if (TheGame.taille_bloc>19)
Bloc.setIcon(new javax.swing.ImageIcon(getClass().getResource("/Anim/3ultra.gif")));

   else if (TheGame.taille_bloc>13)
Bloc.setIcon(new javax.swing.ImageIcon(getClass().getResource("/Anim/2giga.gif")));

   else if (TheGame.taille_bloc>7)
Bloc.setIcon(new javax.swing.ImageIcon(getClass().getResource("/Anim/1super.gif")));

   else
Bloc.setIcon(new javax.swing.ImageIcon(getClass().getResource("/Anim/4null.gif")

// on reprend le tableau créé dans Cjeu
 jeu = TheGame.jeu ;
// Vérification de la fin du jeu
```

```
if (!fini) {
        for(int i = 0 ; i<nbL ; i++) // parcours de tout le tableau
         for(int j=0 ;j <nbC ; j++) {

  // on récupère chaque bouton de la grille
            JButton but =(JButton) jPanelJeuGrille.getComponent((i)*nbC+(j));

  // si la case vaut 0 (donc vide) on enlève l'icône et les bordures
    if ( jeu[i+1][j+1] == 0)  {
        but.setBackground(java.awt.Color.BLACK);
        but.setIcon(null);
        but.setBorder(new javax.swing.border.MatteBorder(new java.awt.Insets(0, 0, 0, 0), new
java.awt.Color(51,51,255)));
            }
  // si la case est différente de 0
    else {
  // Selon la valeur du tableau jeu et du motif choisi, le motif associé est mis en icône
        for(int k=1;k<=nbMotif;k++) {
          if( jeu[i+1][j+1] == k) {
  but.setIcon(new javax.swing.ImageIcon(
   getClass().getResource( "/Anim/"+formeMotif+tailleMotif+k+".gif"  )));
            if(k!=0)
                    but.setBackground(java.awt.Color.BLACK);
                }
  but.setBorder(new javax.swing.border.MatteBorder(new java.awt.Insets(1, 1, 1, 1), new
java.awt.Color(0,0,0)));
                }
            }
          }
    }
  // si le jeu est fini on lance la fonction fin_du_jeu()
    if (TheGame.game_over()) fin_du_jeu();
  }
```

➢ *Enregistrement des meilleurs scores et gestion de la fin d'une partie*

✓ *Enregistrement des meilleurs scores*

A la fin d'une partie, si le joueur à battu le meilleur score dans le mode de difficulté dans lequel il se trouve on lui demande son pseudo puis on l'enregistre ainsi que son score. Il existe trois fonctions utiles à l'enregistrement, à savoir :

- **creation_fichier()** qui crée le fichier s'il n'existe pas et initialise à « personne » le pseudo et à « 0 » les scores dans le fichier *HightScore*.
- **initHightScore()** qui recopie le fichier et l'écrit dans les *JLabels* du *JDialog* des meilleurs scores.
- **sauvegarde()** qui enregistre le score et le pseudo dans le fichier *HightScore*.

✓ *Gestion de la fin d'une partie*

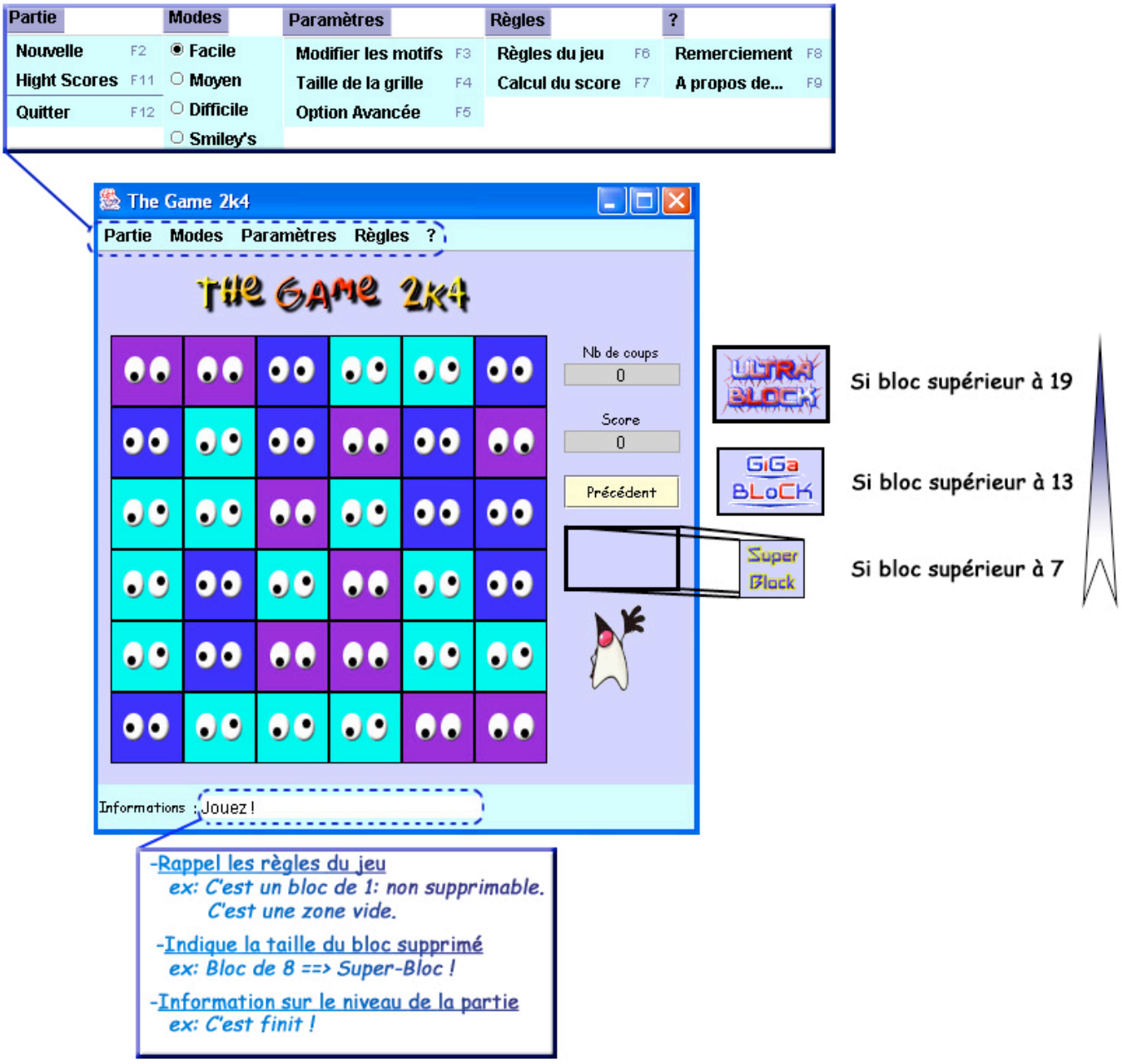
Lorsqu'une partie se termine la fonction **fin_du_jeu()** est lancer. Voici les différentes étapes de cette fonction :

- Affichage du texte « c'est finie ».
- Effet esthétique : le fond (background) de tous les boutons est changé.
- Configuration de la *JDialog* du score.
- Affichage du nombre de blocs restants et des points enlevés en conséquence.
- Calcul du score final.

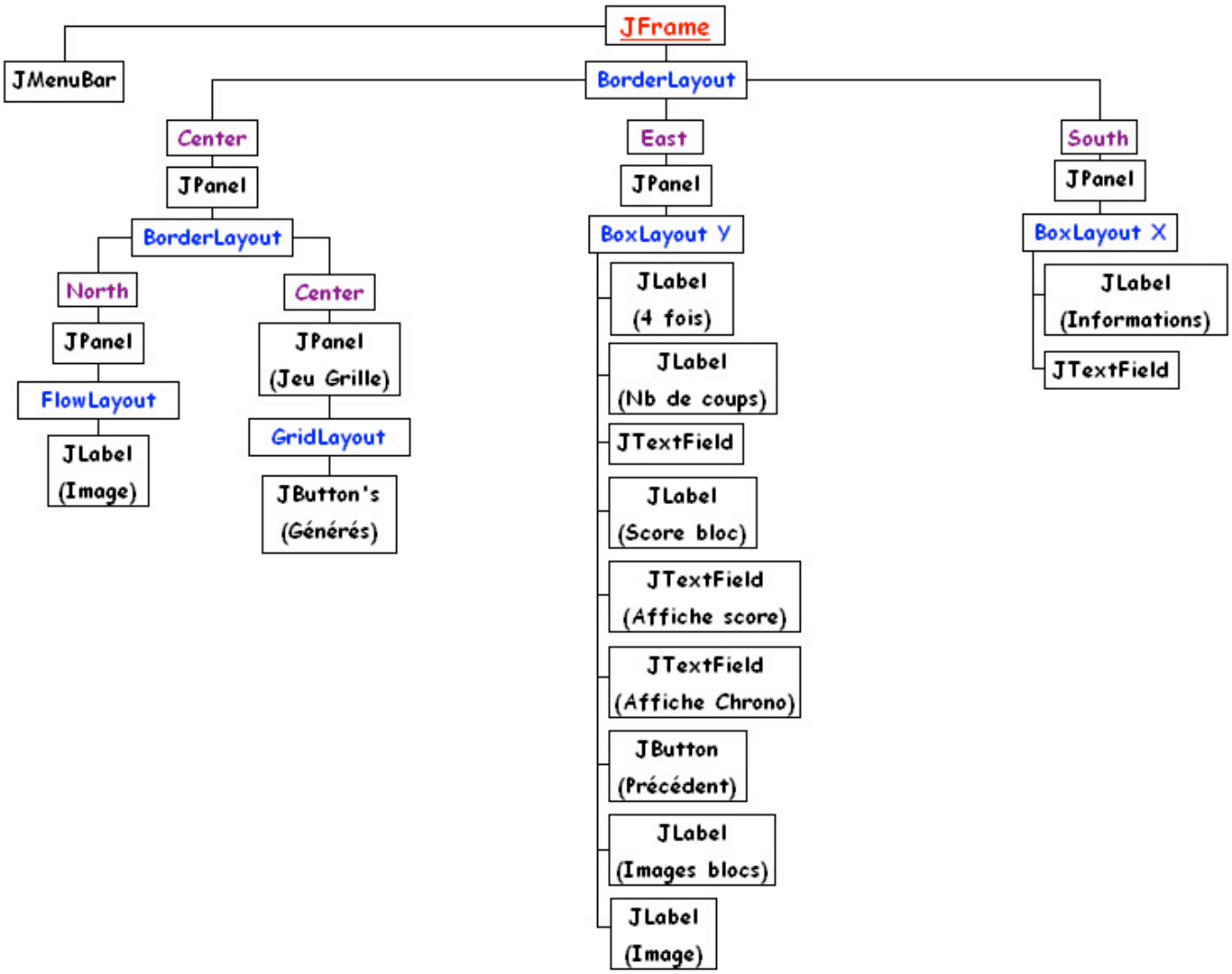
3ème partie : interface graphique

1. Vu d'ensemble

A. Aperçus du jeu à son lancement et aperçus du menu.



B. Stratégie de la fenêtre principale



C. Description des événements

a. Bouton générés automatiquement

L'événement associé à ces boutons permet la récupération de la source du bouton cliqué par la méthode **getSource(),** puis nous récupérons le nom du bouton dans un *int* ce qui nous permet ensuite de trouver les coordonnées du bouton sélectionné grâce à deux formules mathématiques :

```
JButton but =(JButton) evt.getSource() ;
int nb = Integer.parseInt(but.getName());
int coordi = (nb / nbC) ;
int coordj = nb - nbC*coordi ;
```

Nous appelons ensuite la méthode action (qui a été expliquée en détail auparavant) avec comme paramètres ***coordi, coordj***. Puis on lance la méthode **copi_tableau_Cjeu()** (qui a également été expliquée).

b. Bouton Précédent

Ce bouton permet de revenir sur un ou plusieurs coups effectués (en effet le nombre de coups sur lesquels nous pouvons revenir peuvent être définis grâce a un *JDialog*).

Dans un premier temps, nous vérifions si nous avons le droit de reculer : en effet il se peut que l'utilisateur n'ait autorisé qu'un seul coup précédent ou bien que nous soyons revenus au début de la partie. Si tel est le cas nous récupérons le dernier tableau de l'*ArrayList v*, nous le redéfinissons comme tableau courant et nous le supprimons de la liste. Nous effectuons les mêmes opérations pour le score total des blocs afin que celui-ci se décrémente également. Voici le code source de ce bouton commenté :

```
if (TheGame.v.size()>0  && cpt_pre<coups_pre) {
    TheGame.jeu=(int[][]) TheGame.v.get(TheGame.v.size()-1); //on récupère le dernier tableau
    copi_tableau_Cjeu();  // on le définit comme tableau courant
    TheGame.v.remove(TheGame.v.size()-1); // on le supprime de la liste
    TheGame.nbcoup=TheGame.nbcoup-1; //on décrémente le nombre de coup
    AfficheCoups.setText(String.valueOf(TheGame.nbcoup)); //on ré-affiche le nombre de coup
    TheGame.score_bloc_total=  ((Integer)TheGame.score.get(TheGame.score.size()-1)).intValue();   // idem que pour la récupération du dernier tableau, mais ici pour le score
    TheGame.score.remove(TheGame.score.size()-1); // idem
    cpt_pre++; // on compte le nombre de coups précédents qu'il a effectué, ce compteur est mis às zéro lorsqu'il clique sur un motif.
    infos.setText("Vous vous êtes trompé ?!");
    }
    else {
    infos.setText("Impossible de reculer plus...");
    }
```

c. Le menu

- Partie / Nouvelle

Création d'une nouvelle partie, c'est à dire création d'un nouveau **Cjeu** puis appel de la méthode **création_jeu()** (méthode expliquée auparavant).

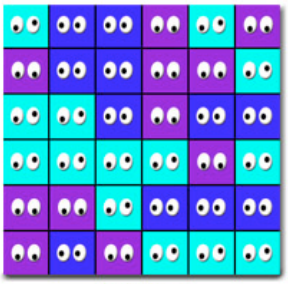
```
TheGame = new Cjeu(nbL,nbC, nbMotif);
creation_jeu() ;
```

- Lanceur de *JDialog*

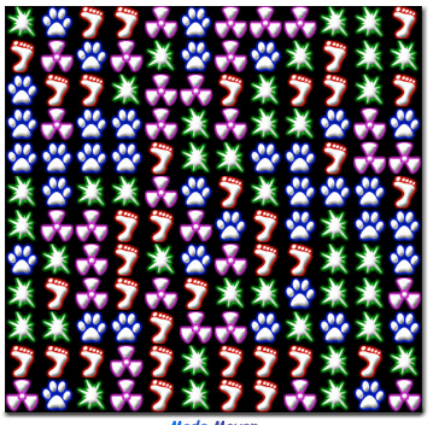
Partie / Hight Score, Paramètres / Modifier les motifs, Paramètres / Taille de la grille, Paramètres / Options avancée, Règles / Règles du jeu , Règles / Calcul du score, ? / Remerciements, ? / A propos de… Se contente de lancer des *JDialogs* par la méthode **show()** ;

- Modes

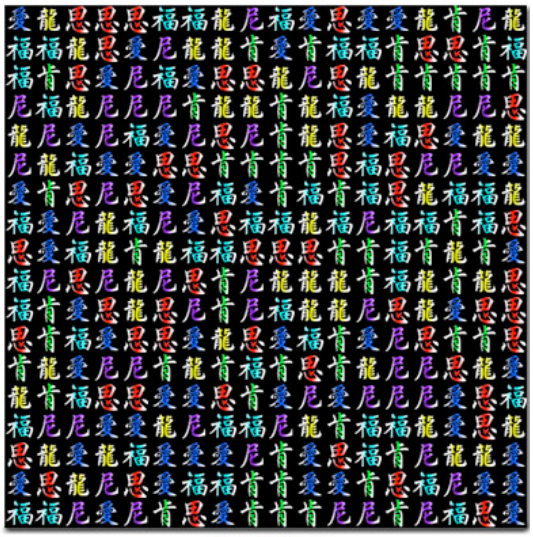
Les différents modes permettent des générations pré-définies de grilles dont voici un aperçu :



Mode Facile
(6 * 6, taille des motifs 50 * 50 pixels)



Mode Moyen
(12 * 12, taille des motifs 40 * 40 pixels)



Mode Difficile
(18 * 18, taille des motifs 30 * 30 pixels)

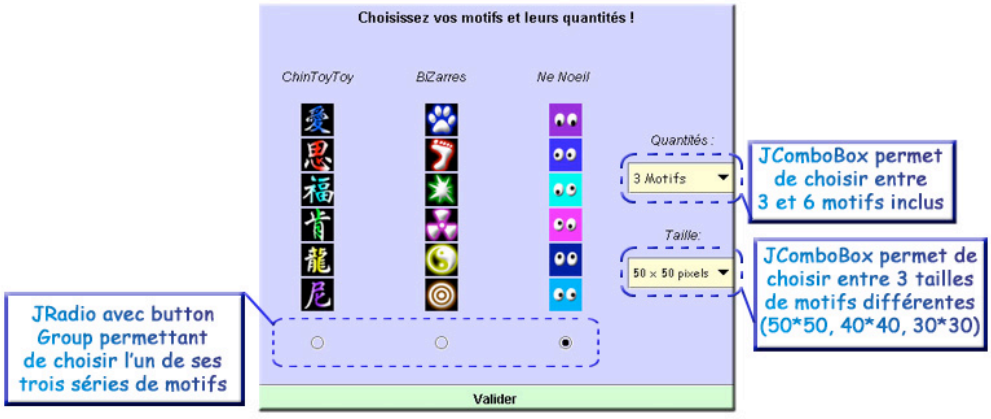
Ces différentes grilles sont générées en recréant un objet de type **Cjeu** avec un nombre de lignes, colonnes et motifs prédéfinis. Voici le code pour le choix du mode facile :

```
nbL=6;  // on définit le nombre de lignes à 6
nbC=6;  // on définit le nombre de colonnes à 6
nbMotif=3 ; // on définit le nombre de motifs à 3
tailleMotif=50; // comme il s'agit d'une petite grille nous pouvons utiliser les plus gros motifs
formeMotif="oei"; // Le choix des motifs est le Ne Noeil
TheGame = new Cjeu(nbL,nbC, nbMotif); // On crée l'objet Cjeu avec nos paramètres
creation_jeu() ; // On génère les boutons et le jeu est interface graphique.
```

2. Panneaux de modifications des paramètres de jeu

A. Choix des motifs

a. Aperçu



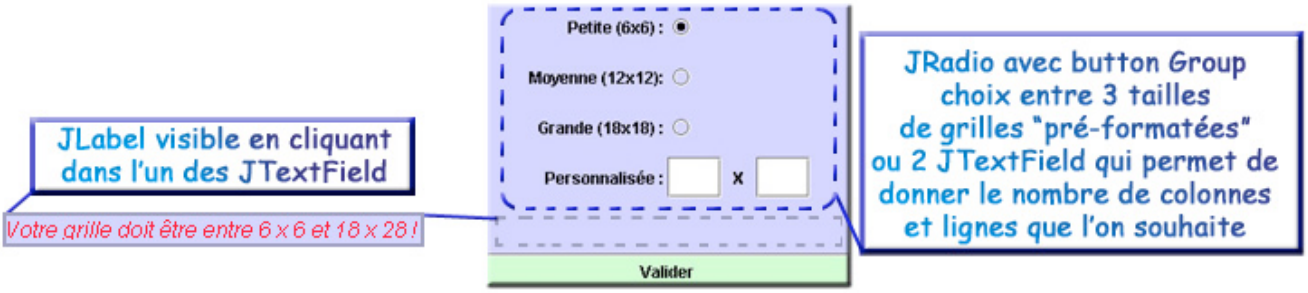
b. Description de l'événement

Seul un événement est généré dans ce panneau, sur le bouton Valider. Il permet de mettre à jour trois variables : ***nbMotif, tailleMotif*** et ***formeMotif***.

Le nombre de motifs et leurs tailles sont sélectionnable grâce à des *JComboBox*, et lors de la validation nous récupérons l'indice de sélection de ces *JComboBox* grâce à la méthode **getSelectedIndex().** Pour le choix de la forme des motifs nous avons un *buttonGroup* qui regroupe les trois *JRadios* et le choix est retourné dans la variable *formeMotif* grâce à la méthode **isSelected()** qui vérifie si un *JRadio* est sélectionné ou non.

B. Taille de la grille

a. Aperçu



b. Description de l'événement

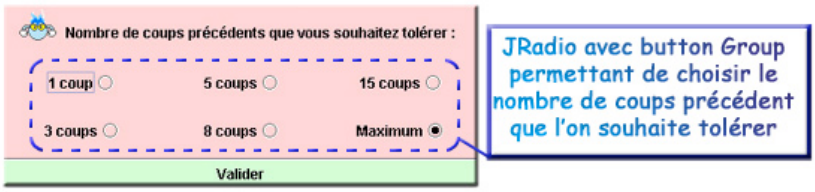
Un événement sur le bouton Valider permet d'obtenir le nombre de lignes et de colonnes que l'ont souhaite avoir. Il y a quatre *JRadios* , trois visibles et un caché et ils sont tous sous le même *buttonGroup*. Lorsque l'utilisateur clique dans l'un des deux *JTextField* le *JRadio* caché se sélectionne ce qui a logiquement pour effet de désélectionner l'un des choix (Petite, Moyenne ou Grande), et un *JLabel* se rend visible ce qui permet l'affichage des tailles maximum autorisées, il deviendra invisible si l'ont re-sélectionne un *JRadio*.

Des vérifications sur le contenu de chaque *JTextField* sont effectuées afin d'empêcher la création d'un jeu dont les tailles ne sont pas comprises dans la plage autorisée ou bien si l'utilisateur entre une lettre, ce qui aurait pour conséquence la génération d'une erreur. Cette vérification parcourt le contenu d'un *JTextField* et vérifie si chaque caractère est compris entre les codes ASCII 48 et 57 inclus , codes qui correspondent aux chiffres. Si ce n'est pas le cas un booléen est mis à *false* et empêchera la validation par la remise du focus dans le *JTextfield*. Nous vérifions aussi si les champs contiennent bien quelque chose ou non…

Si toutes les vérifications sont passées, les variables du nombre de lignes et de colonnes sont définies aux valeurs indiquées ou aux valeurs pré-formatées (en regardant quel *JRadio* est sélectionné par la méthode **isSelected()**).

C. Option Avancée

a. Aperçu

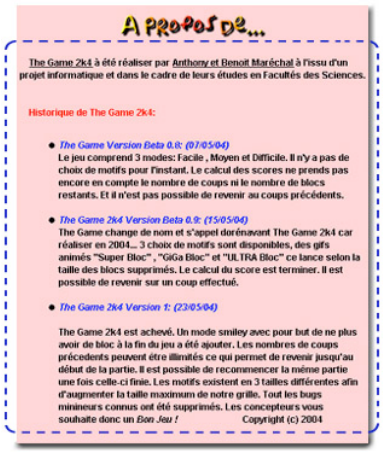


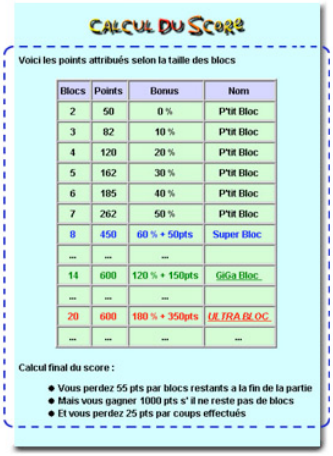
b. Description de l'événement

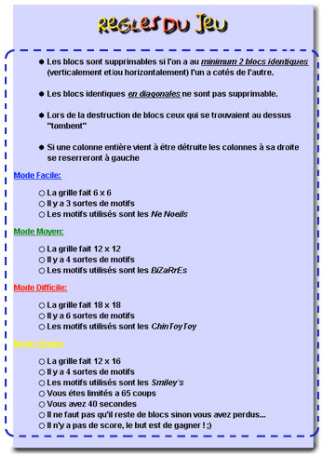
L'événement associé au bouton "Valider" cherche seulement quel *JRadio* parmi les six est sélectionné grâce à la méthode **isSelected()** et re définit la valeur d'une variable qui permet ici de spécifier le nombre de coups précédent tolérés, sachant que « Maximum » permet de revenir jusqu'au début de la partie. Ces *JRadios* font évidement tous parti du même *buttonGroup*.

3. Panneaux d'informations

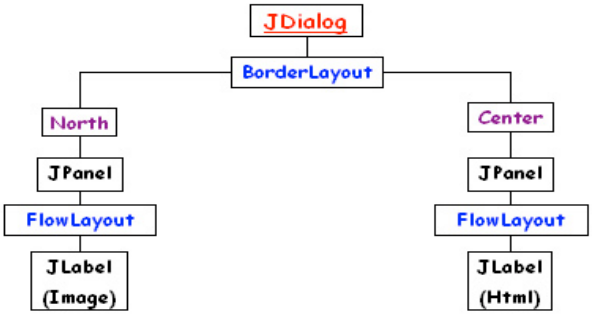
A. Aperçus des panneaux







B. Stratégie des panneaux d'informations

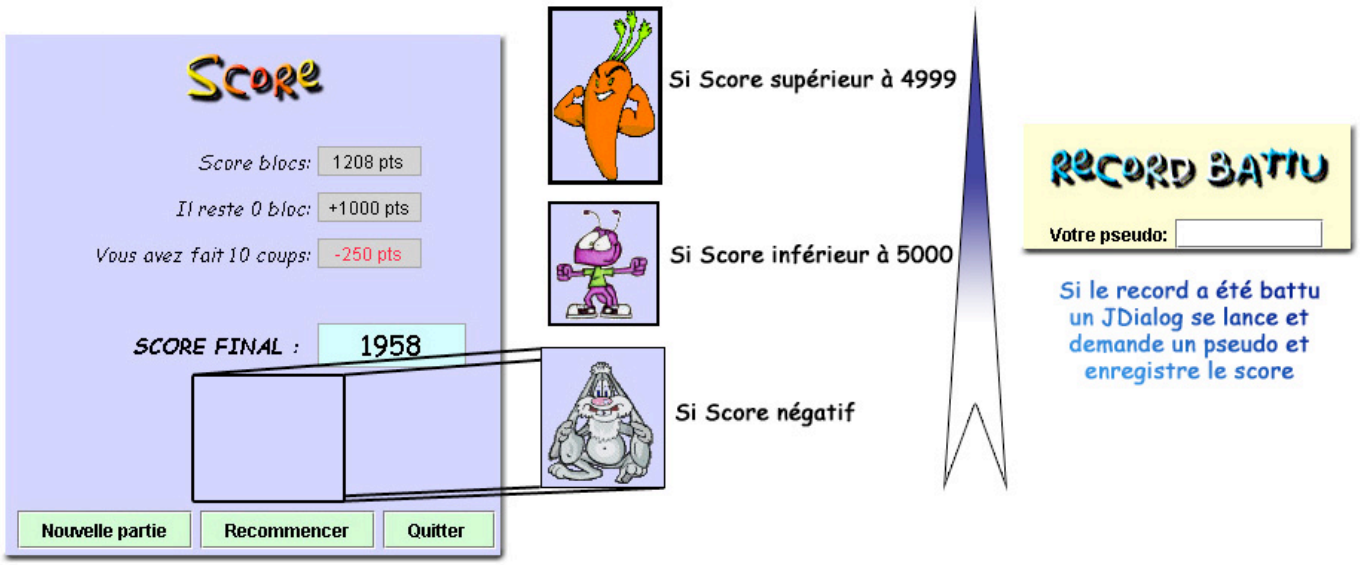


Dans l'aperçu des panneaux, ce qui est entouré en pointillés bleus ce sont les *JLabels* contenant du code *html*. Ce code permet en effet de mettre en place très facilement des tableaux (ex : Calcul du score) ou même des listes à puces (ex : Règles du jeu), c'est pourquoi nous avons décider de créer nos panneaux d'informations en utilisant cette possibilité de structure.

4. Panneaux de score

A. *Panneaux de fin de partie*

a. Aperçus des panneaux



b. Descriptions des événements

- Bouton "Nouvelle partie"

Génère une nouvelle partie en appelant **Cjeu** avec les mêmes variables (ligne, colonne et motif) que la partie en cour.

- Bouton "Recommencer"

Permet de recommencer la partie que l'on vient de finir. Comme nous avons créer une liste de tous les tableaux de jeu dans notre *ArrayList v*, nous nous contentons donc de récupérer le premier tableau généré qui se trouve logiquement à la position 0 de la liste. Puis nous vidons la liste grâce à la méthode **vide_list()** de **Cjeu**.

```
TheGame.jeu=(int[][]) TheGame.v.get(0); // (int [][]) converti l'objet en un tableau à 2 dimensions
TheGame.vide_list();
creation_jeu();
```

- Bouton "Quitter"

Comme tous les boutons "Quitter", la seule instruction est la suivante : system.exit(0) ;

- Panneau Record Battu

Ce panneau se lance si le score de la partie est supérieur au score contenu dans le *JLabel* du High score du mode correspondant. Nous demandons de saisir un pseudo et la validation s'effectue par une pression de la touche "Entrer" (grâce à un *ActionPerformed*). La validation entraîne la réécriture de tout le fichier de sauvegarde et la modification ou l'ajout du score réalisé.

```
p.writeUTF(Pseudo.getText()) ; p.flush() ;
p.writeInt(tt) ;  p.flush() ; // tt est un int qui correspond au score final de la partie.
```

B. *Panneau des High Scores*

a. Aperçu du panneau

b. Description de l'événement

Seul un bouton est présent sur ce panneau, il permet la remise à zéro des scores. Nous réécrivons donc le fichier à des valeurs par défaut (« Anonyme » pour le pseudo et « 0 » pour le score).

5. Mode Smiley's

A. *Aperçu*

B. Descriptions des événements

Les boutons "Nouvelle" et "Quitter" sont les mêmes que ceux du panneau des scores nous n'y reviendront donc pas… Ces panneaux de fin de partie se lancent à la fin du chrono ou au bout du 65ième coup, ou si il ne reste plus de bloc supprimable.

6. Droits d'auteur et autres…

Les images contenues dans le programme sont toutes des gifs animés en libre diffusion trouvés sur Internet. Les images textes (*The Game 2k4*, *High Score*, *Remerciement*, etc…) sont des gifs animés conçus sous Photoshop© et animés par Animations Shop Pro©.

CONCLUSION

Nous sommes content d'avoir pu finir le projet dans les temps, surtout que nous avons quasiment pu mettre toutes les fonctionnalités que nous souhaitions. Les gestions du temps avec les threads nous paraissaient assez intéressantes pour qu'elles fassent l'objet de notre attention malgré que celles-ci soient hors programme. Nous sommes également content d'avoir réussi l'implémentation d'enregistrement des scores , ce genre de fonctionnalité étant indispensable dans la quasi-totalité des programmes et présentant un intérêt évident dans le cadre d'un jeu vidéo.

Nous soulignerons aussi qu'il a été très bénéfique de mettre en place un projet commun pour nos deux matières d'informatique : cela nous a permis de saisir le réel intérêt de la programmation orientée objet avec la gestion d'une interface graphique ce qui était loin d'être évident à priori. Les deux matières nous paraissent extrêmement plus liées maintenant qu'avant la conception du projet.

Après la réalisation de la classe Cjeu, nous avons dû rapidement définir des limites qui n'étaient pas formulées dans l'énoncé du projet, comme par exemple la taille de la grille, pour rendre le jeu plus attractif. Le fait que ces limites ne soient pas énoncées dans le sujet nous a permis de nous rendre compte qu'au delà de l'aspect purement programmation il y a un jeu vidéo qui se doit d'être intéressant. C'est pourquoi nous avons du effectuer des phases de test pour trouver ces limites, ainsi que pour effectuer un calcul « logique » du score.

Quant au travail en binôme, il nous est apparut dans un premier temps comme un handicap : nous avions en effet du mal à nous répartir les tâches. De plus, comprendre un programme qu'une autre personne a écrit peut s'avérer très difficile ; le choix des bons noms de variables, de fonctions, ainsi que la mise en place de commentaires, nous sont apparus comme indispensables par la suite, ce qui nous a permis de nous aider avec une bien meilleur efficacité ! Nous avons réalisé jusque là des programmes dans le seul but qu'ils fonctionnent ; travailler à deux nous a permis de nous rendre compte qu'un programme fini est un programme qui fonctionne mais qui doit également être clair, ce qui n'avait jamais été l'une de nos préoccupations.

Nous avons décidé d'orienter notre projet dans un style cartoon, ce genre de jeux vidéos s'adressant généralement aux jeunes joueurs. Enfin, le langage JAVA et l'E.D.I NetBeans nous semblent bien plus conviviaux et flexibles malgré les préjugés que nous avions pu avoir.

Manuel utilisateur

1. *Nouvelle partie*

Lors du lancement du jeu une grille en mode Facile est créée.
Vous remarquerez, sur la droite de la fenêtre, différentes indications :

- le nombre de coups que vous avez déjà effectués.
- le score réalisé par le bloc que vous venez de supprimer.

Puis dessous ces indications vous disposez du bouton « Précédent » qui vous permet de revenir sur le coup que vous venez de réaliser. Par défaut vous pouvez revenir sur autant de coups que vous le souhaitez, cependant si vous le désirez vous pouvez réduire le nombre de coups précédents tolérés par le jeu en choisissant dans le menu : **Paramètres -> Option avancée** (raccourci clavier : F5) .

En bas de l'application, vous pouvez remarquer différentes informations sur ce que vous venez de faire dans le jeu (par exemple : « Bloc de 18 ==> GiGa-Bloc !! » ou bien encore « C'est une zone vide »).

A la fin de la partie (lorsque plus aucun bloc n'est supprimable ou qu'il ne reste plus de pièce) apparaît le calcul du score. Vous avez alors le choix entre faire une **« Nouvelle partie »,** la **« Recommencer »** ou **« Quitter »** le jeu. Si vous avez battu le record du mode de difficulté dans lequel vous jouez, vous pouvez saisir votre nom ou pseudo. Vous pourrez alors voir votre pseudo ainsi que votre score en sélectionnant dans le menu : **Partie -> High Score** (raccourci clavier : F11).

2. *Choix des différents mode de jeu*

Si vous voulez modifier le niveau de difficulté du jeu, il vous suffit de sélectionner l'un des différents modes de difficulté dans le menu : ***Mode***. Vous avez alors le choix entre Facile (grille de 6x6 avec 3 motifs différents), Moyen (grille de 12x12 avec 4 motifs différents), Difficile(grille de 18x18 avec 6 motifs différents) et enfin Smiley's.

Le mode Smiley's n'est pas un niveau de difficulté mais bien un mode à part entière. Il permet de jouer avec des motifs très particuliers (les smileys), la grille de jeu est fixée à 12x16 et vous ne disposez que de 40 secondes et 65 coups pour qu'il ne reste plus aucun smiley sur la grille ; dans le cas contraire la partie est considérée comme ***« Perdue »***.

3. *Personnalisation de la grille de jeu*

Dans un premier temps vous pouvez choisir les tailles de la grille de jeu (nombre de lignes et de colonnes) dans le menu : ***Paramètres -> Taille de la grille*** (raccourci clavier : F4)

Puis dans le menu ***Paramètres -> Modifier les motifs*** (raccourci clavier : F3), vous pourrez modifier les motifs de la grille, vous aurez alors le choix entre :

- trois séries de dessin.
- le nombre de motifs que vous souhaitez dans une partie (entre 3 et 6).
- la taille qu'ils auront dans le jeu (50x50 pixels, 40x40 pixels ou 30x30 pixels).

4. *Informations complémentaires*

Vous pouvez à tout moment consulter les règles du jeu (raccourci clavier : F6) ainsi que le calcul du score (raccourci clavier : F7) dans le menu ***Règles.***