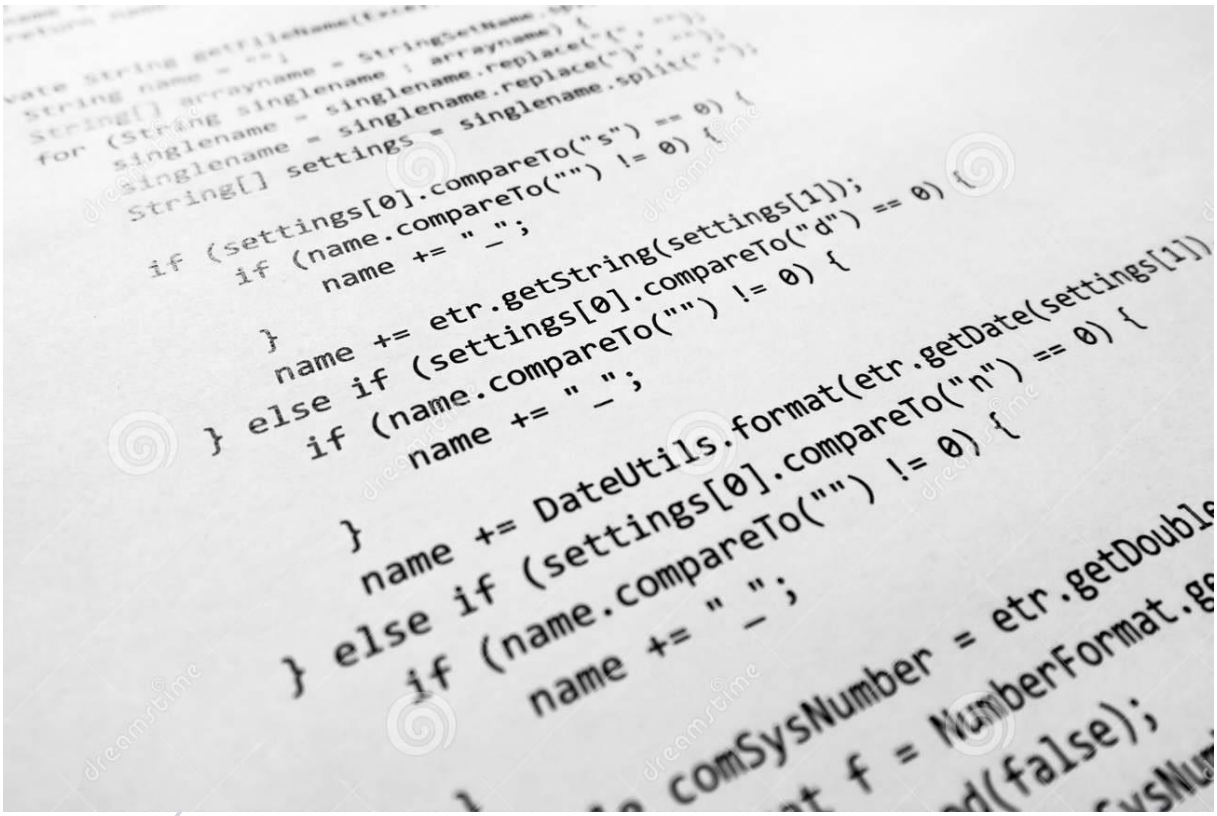
30/05/2016

Projet Java

Le jeu des 6 couleurs



Pierre-Philipe CORDIER
François ROBARD

Table des matières

1. Rappel du sujet 2
Rappel du jeu des six couleurs 2
Version minimale (console) 2
Extensions (Graphique) 2
2. Modélisation (description des classes) 4
Joueur 4
Plateau 5
pileCoord 8
Menu 9
Main 10
Enregistreur 10
MenuListener 11
Sauvegarde 12
PieceImage 12
3. Choix et description des algorithmes utilisés 13
Algorithme principal 13
Règle avancée 14
IA 15
Sauvegarde – Chargement de partie 16
4. Conclusion 17

1. Rappel du sujet

Rappel du jeu des six couleurs

Le jeu des six couleurs se joue sur un plateau comportant des cases hexagonales de six couleurs différentes (rouge, orange, jaune, vert, bleu, violet).

Chaque joueur (de 2 à 4 joueurs) joue tour à tour en choisissant une couleur (autre que la couleur de l'adversaire et sa couleur actuelle). Toutes ces cases sont alors colorées de cette couleur et le joueur prend contrôle de toutes les cases adjacentes à ses cases ayant la même couleur. Si le joueur entoure des cases encore libre de manière à ce qu'elles ne soient plus accessible que pour lui, ces cases deviennent de sa couleur automatiquement. Les obstacles (gris) ne peuvent être capturés par un joueur.

La partie s'arrête quand toutes les cases sont contrôlées. Le gagnant est celui qui contrôle le plus de cases, ou l'lorsqu'un joueur contrôle plus de la moitié des cases au cours de la partie.

Version minimale (console)

Dans cette version tout se passe en mode console, l'enregistrement ne marche pas.

On affiche une couleur par l'initiale de son nom, en minuscule si elle n'est pas contrôlée, en majuscule si elle l'est.

Couleur	Rouge	Orange	Jaune	Vert	Bleu	Violet (indigo)	Obstacle (gris)
Lettre associée	r/R	o/O	j/J	v/V	b/B	i/I	x

Les joueurs entrent le chiffre correspondant à la lettre sélectionnée dans la console (cf. correspondances dans le prochain tableau).

Extensions (Graphique)

La version finale comporte un menu de sélection des options et un affichage graphique grâce à la librairie Swing. Il est aussi possible en cours de partie de l'enregistrer pour la charger plus tard.

Le menu défini le nombre de joueur (2 à 4), s'ils sont un humain ou une IA, et dans le second cas, le niveau de l'IA. Le menu permet de choisir aussi la taille de la grille et la présence d'obstacle ou non. Il est évidemment aussi possible de repartir d'une partie sauvegardée. Il existe aussi un mode partie rapide.

Il existe quatre niveaux d'IA :

- une première prenant une couleur aléatoire
- une seconde qui choisit la couleur qui lui est le plus optimale à ce tour (celle qui fait gagner le plus de points)
- une troisième qui choisit aussi le couleur optimale et en plus dans le cas où il y aurait deux couleurs permettant d'obtenir un maximum de points, choisit celle qui embête le plus l'adversaire.
- Une quatrième qui essaye d'anticiper sur un tour.

La présence de l'IA 3 et 4 peut créer dans certain cas des bugs pour des raisons à ce jour non-identifié.

Il existe une fonction pour capturer les cases qui ne sont plus accessible par les autres joueurs.

L'affichage graphique s'adapte à la taille de l'écran. Il affiche la grille, le tour du joueur ainsi que les couleurs disponible. L'appartenance d'une case par un joueur est matérialisée par un cadre coloré qui entoure l'hexagone correspondant aux cases possédées.

La partie s'arrête quand un des joueurs possède plus de la moitié des cases ou lorsque toutes les cases qui ne sont pas des obstacles sont contrôlées (à améliorer).

Néanmoins, nous avons un problème de compatibilité entre le menu et le déroulement de la partie. Nous n'arrivons pas à trouver le bug. La partie arrive à se lancer mais la présence de la boucle while() dans la fonction Menu.deroulementPartieGraphique() fait planter le jeu. C'est pourquoi nous avons créée deux projets :

-le premier s'appelle menu, il correspond à la version la plus avancé du projet mais nous avons mis dedans la boucle while() fautive en commentaire. De ce fait, il est possible d'initialiser la partie dans le menu qui créera la grille correspondante et l'affichera. Mais la partie n'ira pas plus loin il est effectivement impossible de jouer.

-le deuxième s'appelle projetA1S2 : il correspond à une version où le menu graphique n'est pas finalisé et est désactivé. L'initialisation se fait entièrement dans la console. Cette version permet bien ensuite de pouvoir jouer en mode graphique.

2. Modélisation (description des classes)

On modélisera les couleurs de la manière suivante :

Couleur	rouge	orange	Jaune	Vert	Bleu	Violet	Obstacle (gris)
Chiffre associé	0	1	2	3	4	5	6

On pourra éventuellement rajouter la valeur 6 pour un obstacle.

Classes nécessaires pour réaliser le jeu (avec leur description) :

Joueur

Attributs

- Nom (string) (Indique le nom du joueur, en vue de son affichage graphique)
- Couleur (int) (indique la couleur courante d'un joueur)
- Nb (int) (Indique le numéro du joueur dans la partie, soit 1, 2, 3 ou 4)

Constructeur

- Joueur (string nom) (initialise un joueur et défini son nom)

Méthodes

- set_couleur(int couleur) (Permet de modifier la couleur actuelle du joueur)
- get_couleur() (permet de modifier la couleur actuelle du joueur)
- setNb() (permet de définir le numéro du joueur dans la partie)
- getNb() (retourne le numéro du joueur dans la partie)
- getNom() (Retourne le nom du joueur)
- isIA() (retourne l'attribut isIA qui détermine si le joueur est une IA ou non, ainsi que le niveau de l'IA)
- setIsIA(int isIA) (Permet de définir cet attribut)
- tableJoueur(Plateau grille) (crée un tableau de 2,3 ou 4 cases avec la liste des joueurs dans l'ordre)
- tableIsIA(Joueur[] joueur) (tableIsIA(Joueur[] joueur) (renvoie un tableau qui pour chaque joueur du tableau des joueurs met true si c'est une IA, false sinon).
- couleurIA3v1 (Plateau grille) (Implémente l'IA 3 V1, utilise toute les fonctions restantes de cette classe.)
- couleurIA3v2 (Plateau grille) (Implémente l'IA 3 V2)
- ordreJoueur(Plateau grille) (renvoie un tableau avec en 0 le joueur qui joue, en 1 le joueur du tour suivant, en 2 celui d'après etc)
- nvxPlateau(Plateau ancienneGrille) (créé un plateau copie du plateau actuel mais ou le joueur this à jouer sa couleur optimale par rapport à l'ancienneGrille)
- pointPerdu(int a, Plateau grille, int autre) (renvoie la différence de nombre de point qu'un joueur perd à joueur la couleur en paramètre par rapport à autre en comparaison face à son opti)

- pointPerduAdv(Plateau grilleAutre, Plateau grilleA) (même principe que point perdu mais pour un adversaire)

Plateau

Cette classe contient les informations liées au plateau de jeu et donc nécessaires à son fonctionnement.

C'est dans cette classe que se trouvent la plupart des fonctions du jeu car elles nécessitent souvent un accès facile et rapide à l'ensemble des informations sur l'état du plateau de jeu et sur l'état du jeu lui-même.

Attributs

- nbJoueur (int)) (Indique le nombre de joueurs dans la partie)
- joueur1 (joueur) (permet de définir le premier joueur et d'accéder à ses informations)
- joueur2 (joueur) (permet de définir le second joueur et d'accéder à ses informations)
- joueur3 (joueur) (permet de définir le troisième joueur (s'il existe) et d'accéder à ses informations)
- joueur4 (joueur) (permet de définir le quatrième joueur (s'il existe) et d'accéder à ses informations)
- taille (int) (définit la taille du plateau, c'est-à-dire le nombre de cases pour un côté du tableau)
- typeTableau (int) (définit un tableau carré ou hexagonal, mais cette fonction n'a pas été exploitée)
- Cases (int[][]) (Modélise les différentes cases du tableau et leur état (chiffre de 1 à 6))
- Cases_controle (int[][]) (Modélise les différentes cases du tableau et par qui elles sont contrôlées : 0=personne, 1=joueur1, etc.)
- Fini (boolean) (définit si le jeu est fini ou non)
- obstacle (boolean) (définit si le jeu contient des obsacles ou non)
- aJoue (boolean) (utilisé dans Main.deroulementPartieGraphique() pour savoir si le joueur a deja joue)
- jouantIsIA (boolean) (utiliser dans Main.deroulementPartieGraphique() pour savoir si le joueur est une IA ou un joueur humain)
- jouant (int) (numero du joueur en train de joue)
- veutEnregistrer(boolean) (indique quand l'utilisateur désire sauvegarder la partie)

Constructeurs

- Plateau (Plateau ancienneGrille , int couleurJouee, Joueur, joueurX) (Permet de créer une copie de la grille actuelle puis de lui faire passer un tour où le joueurX jouerais couleurJouee, utilisé pour l'IA3.)
- Plateau (joueur joueur1, joueur joueur2, int type_de_plateau) (création du plateau pour deux joueurs)
- Plateau (joueur joueur1, joueur joueur2, joueur joueur3, int type_de_plateau) (création du plateau pour trois joueurs)

- Plateau (joueur joueur1, joueur joueur2, joueur joueur3, joueur4, int type_de_plateau) (création du plateau pour quatre joueurs)
- Plateau(int nbJoueurs,int typeTableau, int taille, boolean fini,boolean obstacle,int jouant,int[][]cases,int[][] casesControle,Joueur[] joueurs) (Permet de recréer un tableau à partir des informations recueillies à partir d'un fichier de sauvegarde)

Méthodes

- getFin () (retourne l'état du booléen 'fini' afin de savoir si le jeu est fini ou non)
- setFin (boolean fin) (permet de modifier le booléen 'fini', afin d'indiquer le fin du jeu par exemple)
- getJoueur1 () (retourne l'instance de la classe joueur correspondant au joueur1, afin de pouvoir obtenir des informations le concernant ou les modifier)
- getjoueur2 () (idem pour le joueur2)
- getJoueur3 () (idem pour le joueur3)
- getJoueur4 () (idem pour le joueur4).
- getJoueurv2(int numero) (permet d'obtenir un pointeur vers le joueur au numéro indiqué).
- getNbJoueur() (retourne le numéro d'un joueur).
- getObstcle() (retourne la valeur de l'attribut obstacle)
- getCases() (retourne le tableau des couleurs des cases du jeu)
- getCasesControle() (retourne le tableau des possessions des cases du jeu par les différents joueur)
- getVeutEnregistrer() (retourne la valeur de l'attribut veutEnregistrer)
- setVeutEnregistrer(boolean a) (permet de changer la valeur de l'attribut veutEnregistrer)
- getTaille()
- getAjoue()
- setAjoue()
- getJouantIsIA()
- setJouantIsIa(boolean a)
- getJouant() (retourne le numéro du joueur jouant)
- getJouantv2() (retourne un pointeur vers le joueur jouant)
- setJouant(int a)
- getTypeTableau()
- affichageConsole() (Affiche dans la console l'état du plateau à cases hexagonal (lignes impaires décalées vers la droite) à l'aide de lettres. La lettre affichée correspond à la lettre initiale de le couleur, en minuscule si la case n'est pas contrôlée, en majuscule sinon).
- verifControleGlobal() (Cette fonction, utile en début de partie après l'initialisation de la grille, vérifie globalement si un joueur a ses cases adjacentes de la même couleur que ses cases contrôlées. Dans ce cas la fonction remet à jour le tableau de contrôle du jeu afin d'ajouter les nouvelles conquêtes du joueur. Cette vérification est effectuée pour tous les joueurs)
- VerifControle(joueur joueurX) (vérifie si les cases adjacentes au cases contrôlées par un joueur sont de la même couleur, si oui elles sont ajoutées à sa possession)
- VerfiControle2 (joueur_joueurX, int nbJoueur, pileCoord pile, int val) (pour chaque case de la liste fournie, si une nouvelle case de la même couleur non déjà contrôlée est détectée aux alentour, elle est ajoutée à la liste. La fonction analyse toutes les cases fournies initialement, puis toutes les cases ajoutées, jusqu'à avoir détecté l'ensemble du groupe de cases contrôlées par un joueur. Le tableau de contrôle des cases est alors mis à jour)

- joueConsole () (cette fonction en combine plusieurs autres de manière à décrire toutes les étapes nécessaire pour faire jouer un joueur dans la console:

  => l'analyse des couleurs jouables

  => la demande au joueur de choisir parmis ces couleurs

  => L'affectation de cette couleur à ses cases contrôlées

  => La vérification de controle des cases adjascentes aux siennes, et la mise à jour du tableau de controle

  => La vérification permettant de savoir si le jeu et fini ou non

  => Et enfin l'affichage dans la console du plateau)

- couleursJouables() (renvoie la liste des couleurs qu'un joueur peut jouer (sous forme de chiffres associés aux couleurs. C'est à dire toutes sauf celles déjà contrôlées par un joueur.)
- choixCouleurConsole(joueur joueurX, int[] couleurs) (Cette fonction gère l'affichage graphique dans la console du texte demandant le choix de la nouvelle couleur d'un jouer pour le jeu dans la console. On lui fournit le joueur qui joue ainsi que la liste des couleurs jouables. Elle affiche uniquement les informations concernant ces couleurs. Elle permet ensuite au joueur de choisir via la console. Si la couleur choisie est correcte elle est retournée par la fonction. Sinon le choix est redemandé au joueur.)
- estDisponible(int couleur, int [] couleurs) (Vérifie si une couleur est disponible dans une liste de couleurs (de chiffres associés à des couleurs plus précisément). Concrètement : vérifie si un entier est présent dans un tableau d'entiers.)
- changementCouleur (joueur joueurX, int nouvellCouleur) (Remet à jour le tableau 'cases' pour changer toutes les cases contrôlées par un joueur en la nouvelle couleur qu'il a choisi (donnée en paramètre sous forme de chiffre associé)).
- verifFin() (remet à jour le booléen 'fini' associé au plateau si le jeu est terminé, ce qui peut arriver dans deux cas : si un joueur contrôle plus de la moitié des cases ou si toutes les cases non obstacles sont contrôlées.)
- annonceGagnantConsole() (Cette fonction est dédiée à l'affichage dans la console en fin de partie : L'affichage du nombre de points (cases contrôlées) de chaque joueur en face de leur nom, ainsi qui l'affichage du vainqueur.)
- choixCouleurIANiveau1(Joueur joueurX, int[] couleurs) (Implémente l'IA de niveau 1 et lui fait choisir une couleur à jouer parmi les couleurs disponibles qui lui sont fournies en paramètre)
- choixCouleurIANiveau2(Joueur joueurX, int[] couleurs) (Implémente l'IA de niveau 2)
- choixCouleurIAN3v1(Joueur joueurX, int[] couleurs) (Version 1 de l'implémentation l'IA de niveau 3, appel une fonction de la classe joueur)
- choixCouleurIAN3v2(Joueur joueurX, int[] couleurs) (Version 2 de l'implémentation l'IA de niveau 3)
- choixCouleurIANiveau4(Joueur joueurX, int[] couleurs) (Implémente l'IA de niveau 4).
- comptePointsHypothetiquesJoueurParCouleur(Joueur joueurX,int couleur) (Compte le points que gagnerait un joueur en jouant une couleur donnée).
- comptePoints(Joueur joueurX) (Compte le nombre de cases controlées par un joueur)
- regleAvancee() (Permet de changer directement de couleur des cases libres entourées par les cases d'un joueur et ne pouvant plus être capturées que par lui. Permet aussi de transformer

en obstacle toutes les cases libres entourées par des obstacles, de manière a ce qu'il n'y ait pas de cases non capturables)

- reinitialisation(boolean[][] dejaPassee, boolean[] joueursRencontres) (réinitialise les tableau joueursRencontres et dejaPasse à false)
- gererGroupe(int i, Integer j, ArrayList<Integer>[] casesLibres, boolean[][] dejaPassee, boolean[] joueursRencontres) (Par d'une cases d'indice i,j; en determine par recurrence : avec la fonctions versDroite() et versGauche(), le groupe de cases voisines correspondants, qu'on enregistre dans dejaPassee et on enregistre aussi les joueur touchant ce groupe avec joueursRencontres. Puis les enlèves des cases libres à tester et si nécessaire recolore le groupe...)
- enleverGroupe (ArrayList<Integer>[] casesLibres, boolean[][] dejaPassee, boolean[] joueursRencontres) (à ce stade on trouve dans dejaPassee les cases du groupe (valeur true), et dans joueursRencontres les joueurs en contact avec le dit groupe. On enlève à cases libre le groupe vu qu'il a été éxecuté et on change la couleur et les appartenances si nécessaire).
- coteTeste(int a, Integer b) (renvoie le numero du joueur à qui appartient la cases qu'on essaye d'accéder par récurrence (celle en a, b), 0 si libre, 6 si obstacle)
- Integer hg (int i, int j) (fonction renvoyant l'entier à mettre pour j pour aller en haut à gauche quand on se trouve à la ligne i).
- Integer hd (int i, int j) (fonction renvoyant l'integer à mettre pour j pour aller en haut à droite quand on se trouve à la ligne i).
- Integer bg (int i, int j) (fonction renvoyant l'integer à mettre pour j pour aller en bas à gauche quand on se trouve à la ligne i).
- Integer bd (int i, int j) (fonction renvoyant l'integer à mettre pour j pour aller en bas à droite quand on se trouve à la ligne i).
- versDroite (int i, int j, boolean[][] dejaPassee, boolean[] joueursRencontres) (fonction combinée avec la fonction versGauche() afin de passer par toutes les cases d'un groupe par récurrence et de conserver tous les joueursRencontrees du même coup).
- versGauche (int i, int j, boolean[][] dejaPassee, boolean[] joueursRencontres) (fonction similaire à vers gauche)

pileCoord

Cette classe implémente une pile de coordonnées, c'est à dire une pile constituée d'une liste contenant elle-même des listes de deux valeurs correspondant à des coordonnées.

Elle permettra dans le code de stocker des listes de coordonnées pour diverses applications sans avoir à se préoccuper de la taille du tableau les contenant.

Attributs :

- pile_coord (int [] []). (Il s'agit de la pile de coordonnée qui sera manipulée par toutes les fonctions de la classe)

Constructeur

- pileCoord (int [] []) (Permet d'initialiser une pile de coordonnées à partir d'une liste existante).

Méthodes :

- add(int [] coord) (Permet d'allonger un pile d'une valeur et de rajouter la valeur en paramètre à la fin de la pire, quelle que soit sa longueur actuelle).
- take() (Retourne le premier élément de la pile et raccourci la pile d'une valeur en enlevant sa première case).
- read(int val) (Permet de lire une valeur dans une pile à partir de sa coordonnée).
- afficher_pile() (Permet d'afficher la pile dans la console (utile pour les tests de fonctions))
- addVerif(int [] coord) (Permet l'ajout d'une coordonnée supplémentaire à une pile en y plaçant le valeur fournie en paramètre, mais seulement si cette valeur ne se trouve pas déjà dans la liste. Cette fonction sera très utile pour l'analyse des cases contrôlées par chaque joueur)
- getPileCoordLength() (permet d'obtenir la taille d'une pile de coordonnées)
- getPileCoord() (retourne le tableau qui constitue une pile (afin de pouvoir l'analyser dans une autre fonction par exemple))

Menu

Cette classe contient la boucle principale du jeu décrivant son déroulement (en commentaire puisqu'elle bug). Elle contient aussi les fonctions affichant le menu d'initialisation du jeu (choix du nombre de joueur, définition des informations les concernant, définition du plateau de jeu…). Elle permet aussi de mettre en forme la fenetre dans laquelle s'affichera le jeu.

Fonctions :

- main (appel menu())
- iniSwing (permet de mettre en forme la fenetre dans laquelle s'affichera le jeu et de la raccorder au Listener PieceImage)
- deroulementPartieGraphique (fait jouer les joueur tour après tour jusqu'à la fin de la partie (en commentaire car bug), permet de lancer une sauvegarde)
- deroulementPartieConsole (fait jouer les joueur tour après tour dans la console jusqu'à la fin de la partie)
- menu (créer la fenetre et demande entre nouvelle partie et charger partie, créer le lien avec MenuListener)
- menu2 (demande entre partie rapide et partie personnalisé, créer le lien avec MenuListener)
- menu3 (interface de personnalisation d'une partie, créer le lien avec MenuListener)

Main

Cette classe contient la boucle principale du jeu décrivant son déroulement. Elle contient aussi une fonction d'initialisation du jeu depuis la console (choix du nombre de joueur, définition des informations les concernant, définition du plateau de jeu…). Elle permet aussi de créer la fenetre dans laquelle s'affichera le jeu.

Fonctions :

- main (appel jeu())
- jeu (permet de définir si l'on veut charger une partie ou en créer une nouvelle ou jouer en mode console)
- InitialisationJeuConsole (Permet depuis la console de choisir le nombre de joueurs, indiquer leur nom et les créer, créer le plateau)
- iniSwing (permet de créer la fenetre dans laquelle s'affichera le jeu et de la raccorder au Listener PieceImage)
- deroulementPartieGraphique (fait jouer les joueur tour après tour jusqu'à la fin de la partie, permet de lancer une sauvegarde)
- deroulementPartieConsole (fait jouer les joueur tour après tour dans la console jusqu'à la fin de la partie)

Enregistreur

Du fait de la portée des instanciations des classes Fenêtre et PieceImage, la classe enregistreur est pratique pour transmettre ces deux instanciations tout au long du code. Elle permet aussi de transmettre les éléments de "partie personnalisée" au long du code, facilite les problèmes de portée et aide à l'organisation.

Attributs :

- fenetre (Fenetre) (Fenetre créer dans Menu.menu() ou Main.iniSwing())
- PI (PieceImage)
- joueursIni (int[][]) (retient qu'elle joueur est quoi (IA, humain, rien…))
- joueursNoms (JTextField) (retient les noms des joueurs)
- obstacle (boolean) (présence d'obstacle)
- tailleS (JSlider) (pointeur vers le JSlider de Menu.menu3())
- taille (int) (taille de la grille)

Constructeur :

- Enregistreur () (Créé des instanciations des classes fenêtre et une PieceImage).

Methodes : (leur nom est très explicite)

- getPI()
- getFenetre()
- getJoueurIni

- setPI(PieceImage PI)
- setFenetre(Fenetre fenetre)
- setMenuListener(MenuListener tout)
- getJoueursNoms()
- setObstacle(boolean ob)
- getObstacle()
- setTailleS(JSlider js)
- getTailleS()
- setTaille(int t)
- getTaille()

MenuListener

Cette classe permet « d'écouter » l'ensemble du menu et d'agir en conséquence. Il retient dans boutonMenu tous les boutons du menu et ainsi peut les identifiés dans la fonction actionPerformed.

Attributs :

- boutonMenu (AbstractButton[]) (contient tout les boutons du menu dans un ordre bien précis)
- menu (Fenetre) (contient la fenetre qui doit être conserver entre les diffférentes pages du menu et mis à jour)
- piFenetre (Enregistreur)

Constructeur :

- menuListener(Fenetre menu, Enregistreur piFenetre)

Methodes

- getBoutonMenu()
- actionPerformed(ActionEvent e) (repère pour chaque event à quelle bouton du menu il appartient et agit en conséquences)
- isBouttonJoueur(ActionEvent e) (renvoie true si l'event correspond à un bouton entre 5 et 28 de boutonMenu)
- changerJoueurIni (ActionEvent e) (change dans l'enregistreur les paramètres d'initialisation du joueur en fonction du bouton correspondant à l'event)
- demoConsole() (trouve les informations sur les paramètres de la parties personnalisé afin de les affichés dans la console pour vérifier que le menu marche bien)
- lancerPartie() (lance la partie en fonction des paramètres de la partie personnalisée)
- stateChanged() (surveille le JSlider afin d'avoir la bonne valeur pour le paramètre taille de l'enregistreur)

Sauvegarde

Cette classe contient les fonctions de sauvegarde de partie et de reprise de partie depuis un fichier enregistré. Elle contient uniquement des fonctions statiques. Elle existe principalement par question de lisibilité, de manière à regrouper les fonctions liées à la sauvegarde.

Fonctions contenue :

- sauvegarde (Plateau grille, PieceImage PI) (Cette fonction utilise le filedialog pour ouvrir une fenetre Windows qui permet de parcourir les dossiers afin de choisir l'emplacement de la sauvegarde ainsi que le nom du fichier créé. S'il n'y a pas d'erreur elle fait ensuite appel à la fonction EcrireSauvegarde).
- ecrireSauvegarde (Plateau grille, PieceImage PI, String filename, String cheminAcces) (Cette fonction créé un fichier txt contenant toutes les informations nécessaires à une reprise ultérieure de la partie, dans un fichier du nom fourni à l'emplacement fourni grâce à un Buffered writer. La manière dont les informations sont sauvegardées est décrite plus loin).
- ChargementSauvegarde (Enregistreur piFenetre) (Cette fonction utilise le filedialog pour ouvrir une fenetre Windows qui permet de parcourir les dossiers afin de choisir le fichier de sauvegarde à ouvrir. S'il n'y a pas d'erreur elle lance ensuite la fonction lecture fichier).
- lectureFichier (String filename, String cheminAcces, Enregistreur piFenetre) (Cette fonction lit le fichier du nom indiqué à l'emplacement indiqué. La lecture se fait ligne à ligne grâce à un buffered Reader. L'ensemble des données nécessaires à une partie sont alors lues et une partie est alors reprise depuis le fichier de sauvegarde).

PieceImage

Attributs :

- rayon (int) (rayon d'un hexagone)
- Xdepart(int) (abscisse du point en haut à gauche des hexagones)
- Ydepart(int) (Ordonnée du point en haut à gauche des hexagones)
- demiLargeur(double) (demi largeur comme son nom l'indique, ce qui ne correspond pas au rayon car on est pas dans un cercle)
- demiCote(double) (Longueur de la moitié d'un côté d'un hexagone)
- plateau(Plateau) (Plateau à afficher)
- fenetre(Fenetre)
- joueurLabel(JLabel) (contient le JLabel qui affiche le nom du joueur (joueur 1 en indice 0))
- stroke (BasicStroke) (Pour définir le trait utilisé pour faire les contours de l'hexagone)
- strockeWidth (int) (Pour espacer les polygones)
- couleur (Color[]) (La liste des couleurs des hexagones)
- couleurJoueur (Color[]) (La liste des couleurs des contours autour des cases contrôlées)

Constructeur

- PieceImage(Fenetre fenetre, int rayon, int Xdepart, int Ydepart, Plateau plateau)

Methodes

- getFenetre()
- getXdepart()
- getYdepart()
- getJoueurJLabel()
- setJoueurJLabel(JLabel[] a)
- bordJoueur() (Fonction qui dessine les bord des polygones des joueurs. Elle est utilisée par des fonctions comme paint)
- setAttribute(int rayon) (Permet de réinitialiser les attributs à chaque appel de paint afin que la taille de la grille puisse s'ajuster à la hauteur de l'écran).
- paint(Graphics g) (Permet de dessiner la grille (et de l'entretenir à l'aide de la fonction repaint. Elle imprime les polygones en fonction de leur couleur et dessine leur contour en fonction de leur appartenance à un joueur)
- actionPerformed(ActionEvent e) (Lié au bouton d'enregistrement dans la barre de menu pour lancer une sauvegarde).
- MousePressed(MouseEvent e) (Permet d'écouter les entrées de la souris pendant la partie afin de déterminer la couleur que le joueur jouant choisit et si il peut jouer (couleur autorisé, tour d'un humain…)
- mouseEntered(MouseEvent e)
- mouseExited(MouseEvent e)
- mouseReleased(MouseEvent e)
- getRayon()
- itemStateChanged(ItemEve,t arg0)

3. Choix et description des algorithmes utilisés

Quelques fonctions ont un code court et un fonctionnement simple qui se comprend immédiatement à la lecture. D'autres sont plus complexes et nécessitent une explication. Seules celle si seront explicitées.

Algorithme principal

Selon la version dont on parle la main appelle soit Main.jeu() soit Menu.menu().

Dans les deux cas, il s'agit d'abord de créer une grille selon les exigences du joueur puis grâce à Menu/Main.iniSwing() de créer ou de réinitialiser la Fenetre qui contiendra le jeu. C'est à ce moment précis qu'on créera l'objet PieceImage qui permettra, n'ont seulement de dessiner la grille grâce à PieceImage.paint(), mais aussi d'écouter les inputs du joueur liée à la partie en cours (le mouseListener pour les coup joué et l'actionListener pour le menuBar). Une fois tout ceci initialisé est alors lancé la fonction centrale du jeu : Menu/Main.deroulementPartieGraphique(). C'est cette fonction qui contient la boucle while() qui à chaque occurrence de la boucle fait jouer quelqu'un (IA ou humain). On peut à tout moment de cette boucle activer la fonction de sauvegarde contenue dans la classe Sauvegarde.

Cette boucle prend fin quand l'attribue Plateau.fini passe à true. C'est attribue peut à chaque fois qu'un joueur joue passé à true si les conditions de fin d'une partie son respecté. On active alors la fonction de fin de partie : Plateau.annonceGagnantConsole() .

Il me parait important de préciser qu'à chaque occurence de la boucle soit c'est le tour de l'IA et on appelle la fonction Plateau.joueIAGraphique() soit c'est au joueur et l'on attend que le mouseListener soit activé à juste titre et c'est à ce moment que le joueur joue notamment grâce à la fonction Plateu.joue().

Enfin, il faut différencier l'initialisation parlé dans le début de cette rubrique en fonction des deux possibilités :

-on a appellé Main.jeu(). Dans ce cas la grille est initialisé dans la console grâce Main.initialisationJeuConsole() ou une partie charger avec Sauvegarde.chargementSauvegarde() et la Fenetre est créer dans iniSwing() .

- on a appellé Menu.menu(). La grille est initialisé par le passage successif dans Menu.menu(), Menu.menu2() puis Menu.menu3() et la logique associé qui se trouve dans MenuListener. Tous les éléments relatif aux choix du joueur ainsi récupérer sont envoyé dans l'Enregistreur nommé piFenetre qui contient comme son nom l'indique notamment la Fenetre qui a été créer dans Menu.menu() pour la réutiliser dans Menu.iniSwing.

Règle avancée

Cette fonction (Plateau.regleAvancee()) a pour but de convertir de la couleur voulue les cases encerclées par un joueur (ou un obstacle). Il en existe deux version dont la plus optimiser est celle utilisé (càd, pas en commentaire). Son concept est de dresser une liste de toutes les cases pas encore capturer qui ne sont pas des obstacles, dites cases libres, et ensuite pour chaqu'une d'entre elle (sauf si elle a été retiré de la liste précédemment lors de la fonction) on dresse l'ensemble de ces « voisines » (dejaPassee) ainsi que des joueursRencontres pour créer un groupe grâce aux fonction Plateau.versDroite() et Plateau.versGauche(). Ce groupe de cases voisines sera changer de couleurs si besoin est dans Plateau.gererGroupe() puis supprimer de la liste des cases à tester dans Plateau.enleverGroupe().

Ainsi on obtient un coût de :

Soit n le nombre de cases libres, et l le nombre de cases libres

Coût = l * (reInitialiastion() + enleverGroupe() + gererGroupe()) + n* (versDroite() ou versGauche()) + regleAvancee()

On peut remarquer une très nette progression de la rapidité de deux IA à jouer l'une contre l'autre notamment dans le cas le plus critique : le début de partie. Cependant, le nombre d'appel récursif peut poser problème quand la grille dépasse 60 cases de taille. En-effet le nombre d'appel récursif au premier tour est de l'ordre de taille².

IA

Nous allons procéder à une description du mode de fonctionnement des différentes IAs.

- IA 1 :

Elle choisit une couleur au hasard parmi les couleurs jouables qu'on lui fournit.

- IA 2 :

Elle compte le nombre de points qu'elle pourrait gagner pour chaque couleur jouable et choisi la couleur qui lui rapporte le plus de points. Dans le cas ou deux couleurs apportent un maximum de points, elle choisit le première couleur qu'elle a trouvé apportant ce maximum de points.

Afin de calculer le nombre de points que rapporterait de joueur une couleur à un joueur, la fonction utilise la fonction « comptePointsHypothetiqueJoueurParCouleur » qui recréé une copie du tableau, y fait jouer la couleur qu'on lui donne en paramètre au joueur donné en paramètre avec les fonctions de jeu déjà existantes, et retourne la différence de points engendré en jouant cette couleur.

- IA 31 et 32 :

L'IA 31 et 32 ne sont que deux versions de la même IA. Nous ne développerons pas les détails de fonctionnement de cette IA dans la mesure où un bug persiste sur lesquelles nous n'avons pas eu beaucoup de temps à consacrer.

Son concept est de testé toutes les couleurs disponibles au tour de l'IA et d'y attribué un nombre « total ». La couleur choisie au final sera celle avec le plus grand total. Pour calculer le total d'une couleur a, on fait la différence entre les points que perd le joueur à jouer a plutôt qu'une couleur « autre » à la moyenne des points perdu par les adversaires s'il ne peuvent pas jouer a en anticipant successivement les coûts des autres joueurs sur un tour.

- IA 4 :

Cette IA est une amélioration de l'IA 2. Elle joue exactement de la même manière sauf que quand deux couleurs ou plus permettent de gagner un même maximum de points, alors elle essaye de choisir parmi ces couleurs celle qui embête le plus un adversaire pour le tour en cours, de manière à le ralentir au maximum.

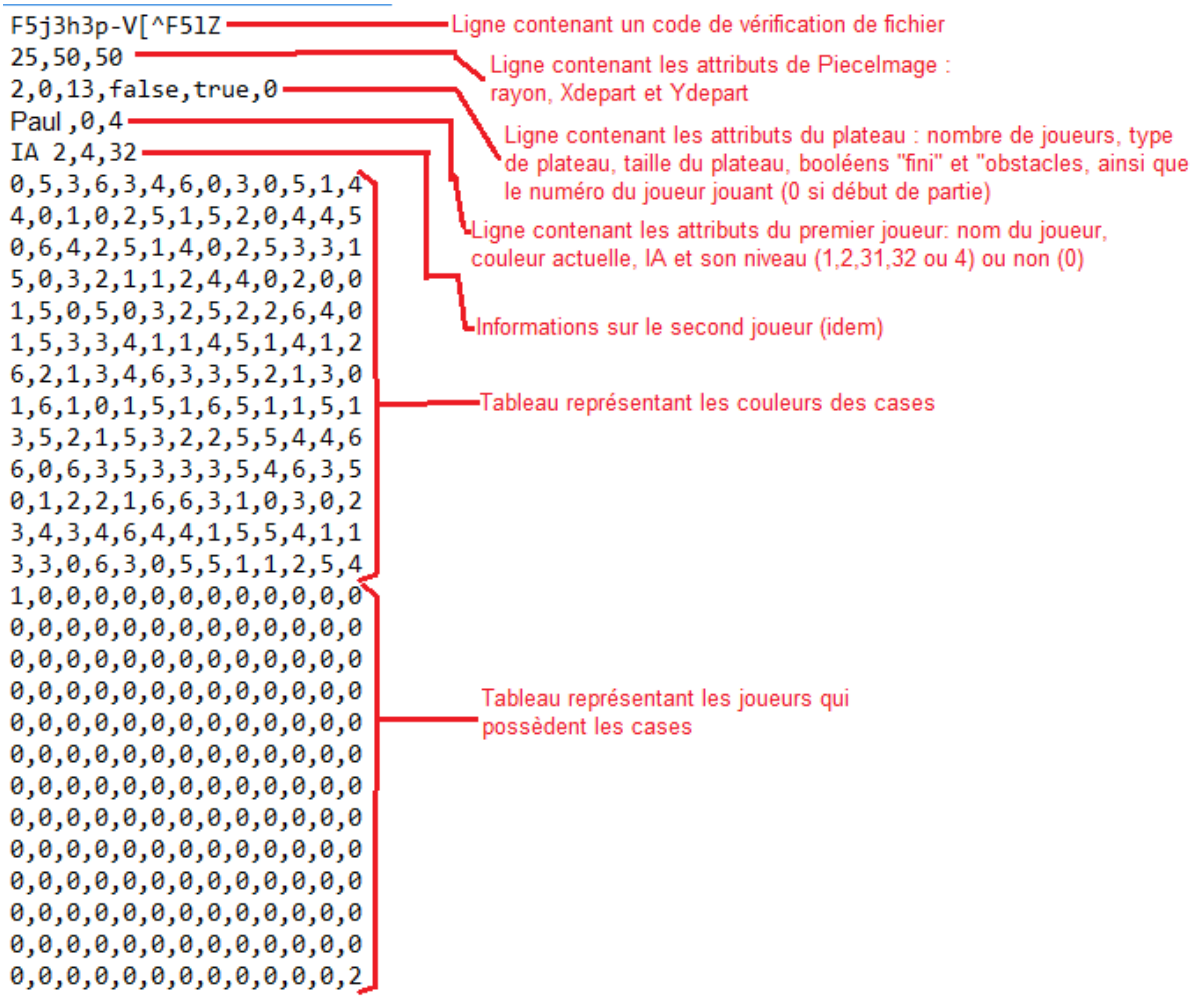
Pour cela elle détermine la liste des couleurs permettant d'obtenir un même maximum de points. Elle calcule ensuite le maximum et le minimum de points que peut gagner chaque joueur adverse avec les couleurs de cette liste. Elle choisit le joueur pour lequel l'écart entre le maximum et le minimum est le plus grand possible : c'est le joueur qu'on embêtera et ralentira le plus en choisissant sa meilleur couleur. La fonction retourne donc parmi le tableau de couleurs amenant le maximum de points celle qui pourrait rapporter le plus à l'adversaire choisit.

Suite à un ensemble de tests on se rend compte que cette IA prend une avance significative par rapport à l'IA 2 et qu'elle est beaucoup plus forte.

Sauvegarde – Chargement de partie

On active la sauvegarde en appuyant sur le bouton « file » puis « sauvegarder » lors de la partie graphique. L'attribut « veutEnregistrer » de la classe plateau passe alors à la valeur « true ».

Dans la boucle while principale du jeu se trouve une boucle if de sauvegarde qui est activée lorsque l'attribut précédent vaut « true ». On fait alors appel à la fonction « sauvegarde » qui permet de choisir le nom et l'emplacement du fichier txt de sauvegarde. Elle fait ensuite appel à la fonction ecrireSauvegarde qui écrit les informations dans le fichier txt selon l'organisation suivante :



Le code de vérification en début de fichier permet de vérifier qu'on ouvre bien un fichier de sauvegarde et pas un fichier txt quelconque qui engendrerait une erreur.

Le nombre de joueurs stocké peut aller de 2 à 4 et la taille des grilles sauvegardées est variable et permet de stocker toute tille de grille.

Ce type de sauvegarde permet de reprendre une partie en cours au joueur qui devait jouer au moment de la sauvegarde.

L'attribut sauvegardé « type de Plateau » devait être utilisé pour définir la forme des cases mais comme elles ne peuvent finalement être qu'hexagonales il n'a pas vraiment d'intérêt. Il pourrait être utile pour un développement ultérieur du jeu.

4. Conclusion

Bien que nous n'ayons pas réussi à debugger le menu, le décompte de fin est imparfait et certaine IA bugs encore ; nous pensons avoir réussie à atteindre un jeu satisfaisant. En-effet, il nous a permis d'améliorer nos compétences en Java tout d'abord. Notamment pour ce qui est de la bibliothèque Swing que nous ne connaissions pas avant. Mais aussi à réussir à s'organiser car contrairement à la création de site internet il était ici beaucoup plus compliqué de répartir le développement en page parfaitement indépendantes.

De plus, nous avons réussi à satisfaire la majorité des exigences du projet et le jeu est globalement fonctionnel et propre. Nous aurions aimé corriger les imperfections, peut-être d'ici la présentation…