Rongeard Benoit, Raton Julien,
Bouquet Antoine, Da Silva Campos Anis

G4S3

Compte rendu Mastermind

Sommaire

1) Organisation

a. Les règles

Le Mastermind est un jeu qui consiste à faire deviner à un adversaire une certaine combinaison de couleur, il se joue à deux joueurs, un qui effectue la combinaison à faire deviner, et l'autre qui devra trouver la combinaison établie par le premier.

L'interface principale propose trois choix différents qui sont « jouer », « option » et « Quitter ». Le choix option est le plus important il permet de paramétrer le jeu, et ainsi faire une partie personnaliser (Ordi VS Ordi, Joueur VS Ordi, Joueur VS Joueur et choisir le nombre de couleur différentes). Une fois les options choisis, il suffit de lancer le jeu avec le choix « jouer », on remarquera que l'affichage est très agréable car l'ergonomie est bien adaptée à la console Eclipse.

Le jeu affichera une boule blanche pour une boule de couleur présente, mais mal placée, et une boule rouge pour une boule de couleur présente, et bien placée.

Le jeu se termine lorsque le joueur (ordi ou humain) trouve la combinaison, le joueur devineur sera alors gagnant, ou lorsque le nombre de possibilité maximum a été atteint, le joueur devineur sera alors perdant.

b. Repartition des tâches

Les tâches ont été réparties de manière à que tout le monde travaille sur chacune d'entre-elle. Cependant, nous avons pu observer que chacun d'entre nous à plus travailler dans certains secteurs comme suit :

Anis :
- Imagination de l'algorithme
- Structure du code
- Rédaction du compte rendu

Julien :
- Imagination de l'algorithme
- Structure du code
- Rédaction du compte rendu

Antoine :
- Mise en place des divers menus
- Rédaction du compte rendu

Benoit :
- Rédaction du compte rendu
- Mise en place des divers menus

c. Répartition du temps de travail

Ensemble	Nom Fonctionnalité	Valeur Marchande	Temps estimé	Temps Passée	Priorité
Règles:	Nb pièces à placer (au choix)	200	1	1	2
6h	Dupliquer les couleurs (au choix)	200	1	1	2
	Nb de coups (compter et fixer)	100	1	1	2
	Nb de couleurs dispo (fixe)	100	1	1	2
	Position des marqueurs (non positionné)	100	1	1	2
Modes:	Algorithme Maison	(Ordi Vs Ordi)	tout le long	6	1
21h	Ordi Vs Ordi	500 + 2000	16	9	1
	Humain (devineur) VS ordi	200	2	2	5
	Ordi (devineur) VS humain	400	3	3	3
	Choix possible	50	1	1	4
Interface:	Interface texte	1000	1	1	1
2h	Interface texte + fichier config	200	0	1	2
	Interface texte interactive	400	1	1	1
	Interface graphique	200	0	0	
Rapport:	rapport + tâches administrative		3	3	1
3h					

Chaque tâche a été effectuée par l'ensemble de l'équipe, elles ont été partagées par des sous-tâches permettant une gestion du temps optimale afin de rendre le projet dans les temps.
De plus, l'intégralité des tâches ont été effectué par l'équipe sans prendre en compte d'aucune source extérieur. De ce fait, l'algorithme, qui était une des tâches la plus compliqué et longue à effectuer, a été faite dans les temps.

d. Algorithme retenu

L'algorithme de l'ordi devineur est : il créé la liste des possibilités en fonction du nombre de boules et de couleurs. Il propose alors au hasard une possibilité de la liste. Quand le jeu lui renvoi le nombre de pions rouges et blancs, il va comparer la proposition avec toutes les possibilités de la liste en supprimant toutes celles qui n'ont pas le même nombre de pions blancs et rouges. De ce fait le nombre de possibilités est réduit en tendant vers la solution unique. L'algorithme prend en moyenne 5 tours pour trouver la combinaison.

2) Choix des outils

Notre choix d'IDE s'est tourné sur Eclipse : effectivement, Eclipse propose une auto complétion qui est beaucoup plus intuitive que celle de NetBeans (de notre point de vue). De plus certains de ses outils sont plus faciles d'accès et plus intuitif pour nous. De plus Eclipse comme NetBeans propose une gestion intégré vers GitHub.

Eclipse, grâce à ses multiples outils, dont plusieurs que nous avons découvert pendant le projet, nous a été d'une grande aide, et d'un grand soutien lors des étapes nécessaires au développement (refactoring, débogage,…).

3) Refactoring

Après avoir fini de taper le code, nous avons utilisé les outils à notre disposition pour le refactoring par Eclipse, c'est-à-dire : clic droit sur le projet → Propriétés → Java Code Style → Formater. Ici nous avons créé un nouveau modèle, suivant les demandes du client, à savoir :

- Les accolades des classes sont sur la ligne suivante.
- Les accolades des méthodes sont sur la ligne suivante, avec une demi-indentation.
- Les "else" passent systématiquement à la ligne.
- Indentation de 4 caractères.
- Dans les commentaires, les descriptions de paramètres sont alignées.

Cependant l'outil de refactoring d'Eclipse ne permet pas de faire la demi-indentation avant une accolade de méthode. (Jusqu'ici, c'est le seul souci que nous ayons rencontré sur l'IDE Eclipse)

En refactoring supplémentaire, nous avons fait en sortes que le code soit plus joli :

- Indentation des paragraphes entre accolades (les « if », les « while », les « for », etc.)
- Suppression des accolades inutiles pour les « if » qui tiennent en une ligne (avec une seule instruction)

Après refactoring, le code est plus lisible, et bien plus aéré.

4) Génération de la Java Doc

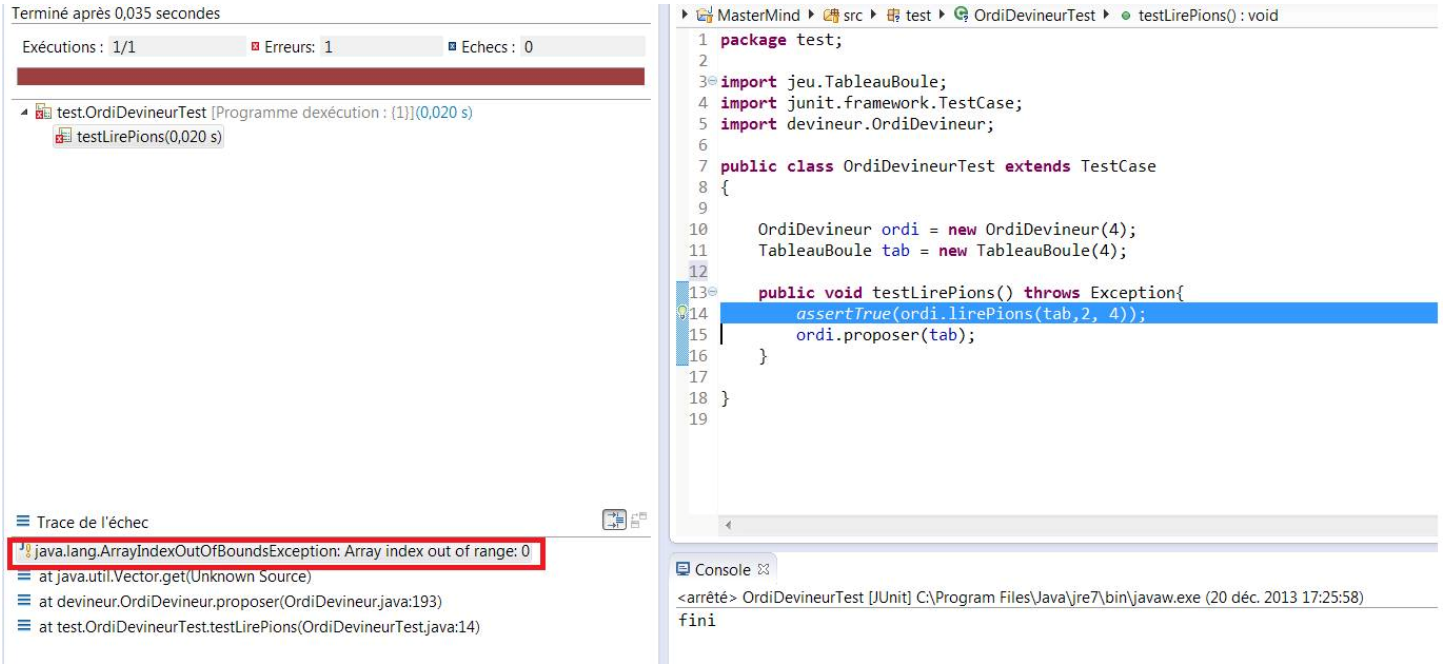
Pour générer la Java Doc, Eclipse propose un plug-in qui se nomme JAutodoc : clic droit sur le projet → JAutodoc → add Javadoc. L'outil est très simple d'utilisation, cependant il est très peu précis, il nous a fallu compléter la plupart de la Documentation générée.

Nous avons entièrement créé la Java Doc de la classe OrdiDevineur, après avoir généré la Java Doc automatiquement pour tout le projet.

5) Tests

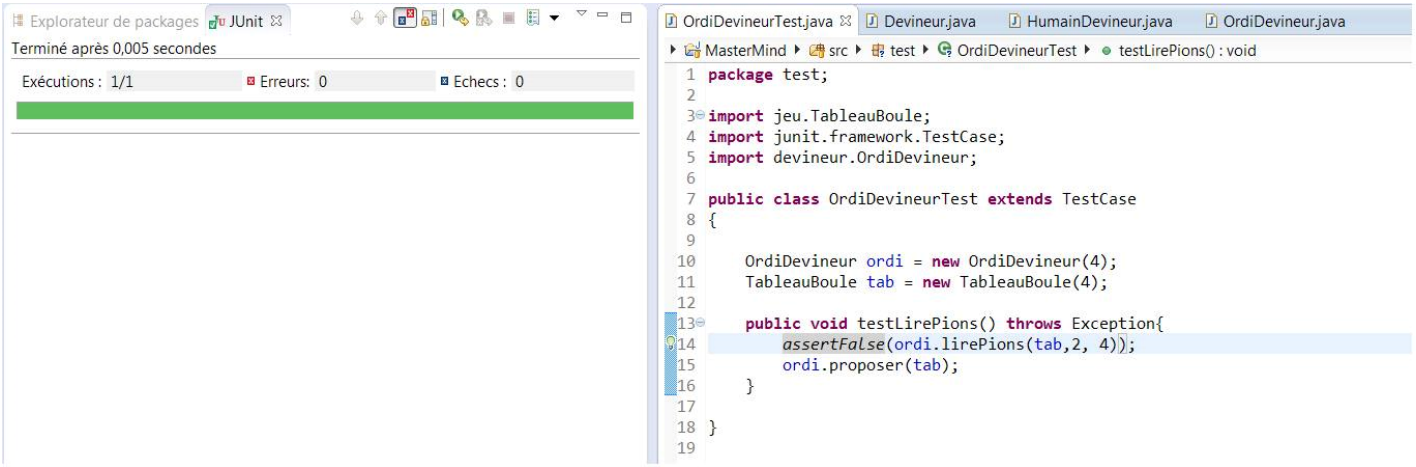
Quels tests ? Quels jeux ? Comment les construire ? Comment les executer?

Pour les tests, nous avons choisi de tester si le nombre de pions (blancs et rouges) sont supérieur au nombre de pions total : le test donne un tableau de boules quelconque avec un nombre de pions volontairement faux. Cela aura pour conséquence de vider la liste des possibilités. Voir photo ci-dessous :



A ce stade là le programme plante : la liste de possibilité étant vide, l'algorithme essaye d'accéder aléatoirement à un indice, qui n'existe pas.

Il a donc fallut reprendre le code, et le corriger, et après correction, l'algorithme passe les tests :



Les changements effectués sont dans la méthode lirePiont de la classe OrdiDevineur :

On ajoute alors un test pour la conformité du nombre total de pions blancs et rouges confondus (qui doit être inférieur ou égal au nombre de boules).

6) Bilan

a. Problème techniques

- Problème pour « rafraîchir la console » pour l'affichage du menu et du jeu.. en effet on ne peut pas effacer le contenu de la console d'Eclipse. La solution à ce problème a été de sauté des lignes à chaque moment où on en a besoin dans les affichages, pour un rendu de l'interface texte plus lisible.

b. Problèmes organisationnels

- Au niveau du temps, nous devions quand même avancer chez nous pour continuer le projet, car les heures de cours ne suffisait pas pour avancer sur certaines tâches (ex : trouver l'idée de l'algorithme, finir le rapport).
- Les sous-tâches étaient au final mal organisé : certains d'entre nous avaient des tâches qui demandaient beaucoup plus de travail que d'autre tâches, il fallait donc s'entre-aider continuellement tout au long du projet.