Projet de programmation

Bilan

Shing Shang

(Java)

Valentin Bossard
Alexis Melo da Silva

DUT Informatique 1ère année
TP2.2

Sommaire

Introduction

Le but du projet était de proposer une version informatique du jeu de stratégie Shing Shang. Nous avions déjà réalisé ce projet dans le langage de programmation C, désormais le langage imposé était le Java, qui à la différence du précédent est un langage orienté objet.
Le projet était à réaliser en binôme, le dossier ci-dessous permettra de rendre compte des étapes de la réalisation du projet, du fonctionnement et du déroulement d'une partie de jeu normale, ainsi que notre ressenti sur ce que nous a apporté ce projet et sa réalisation.

I. Fonctionnalités implémentées

A. Fonctionnalités de base

- Affichage du plateau de jeu
- Placement de l'armée de chaque joueur sur le plateau
- Possibilité de jouer en multijoueur local
- Lancement d'une nouvelle partie
- Création de joueurs
- Choix du joueur qui commence la partie
- Détection automatique de la fin de partie

B. Fonctionnalités ajoutées

- Un menu permettant de choisir quelle partie on veut charger (continuer)
- Sauvegarde automatique de la partie
- Possibilité de supprimer une partie
- Possibilité de choisir au hasard le premier joueur
- Statistiques de fin de partie (Statistique de la partie et des joueurs : Nombre de sauts, nombre de tours, nombre de Shing Shang)
- Consultation des règles du jeu depuis le menu

II. Organisation du programme

A. Choix de conception (Découpage du programme)

Etant donné que la multitude de diagrammes de classe auxquels nous aurions pu nous référer nous avons dû effectuer des choix de conception que nous allons expliciter afin de mieux comprendre notre code.
Tout d'abord nous sommes parti du diagramme de classe de monsieur Zimmerman (cf : Annexe n°1) pour générer toute la structure du programme (classes, liens entre classes, attributs, méthodes…) auquel nous avons apporté quelques modifications.
Nous avons d'abord supprimé toutes les classes permettant d'étendre le projet partant du principe que nous devions d'abord réaliser le minimum et avoir un programme qui marche, puis cherche à l'étendre.
Donc toutes les méthodes incluant IObservateur et l'héritage entre JoueurAbstrait et Joueur a disparu.
Pour essayer de respecter les bonnes pratiques de programmation et ne pas surcharger le main nous avons aussi ajouté une classe " GestionPartie " comportant toutes les méthodes relatives au menu principal et à la gestion d'une partie, son seul attribut est une Partie (celle en cours) et par conséquent nous avons créé une autre énumeration "ActionsMenu" qui liste toutes les actions possible depuis le menu principal.
On a rajouté des méthodes getter qui prennent en paramètre une abréviation dans ces énumération, afin de retrouver une action/direction à partir de la saisie de l'utilisateur.

B. Choix de conception (Règles du jeu)

Il existait également des incohérences et imprécisions quant aux règles données par les professeurs (elles pouvaient varier d'un professeur à un autre), voici alors des précisions sur les règles que nous avons appliquées à notre jeu (en plus de celles de bases et admises de tous).

Pendant le tour d'un joueur le premier saut ne compte pas, c'est à dire que peu importe si le saut est effectué sur un Bushi adverse ou allié, le Bushi sauté ne disparaîtra pas. C'est seulement à partir du deuxième saut que faire un Shing Shang sur un bushi ennemi entraîne la disparition du pion sauté. Il est autorisé de sauter autant de fois que l'on souhaite par dessus des Bushis alliés, le changement de tour s'effectue seulement après une action "Glisser" ou en choisissant l'action "Fin". Il est possible de rejouer après un Shing Shang, mais seulement avec un autre Bushi. Il est impossible de passer son tour. Après un saut, il est impossible de retourner directement sur la case d'origine.

III. Répartition des tâches

A. Tâches communes

Tout d'abord nous avons reproduit le diagramme de classe avec argoUML et nous avons généré les classes et prototypes des méthodes automatiquement. Nous avons supprimé toutes les méthodes jugées "inutiles" comme expliqué précédemment et nous avons commencé à coder ensemble. Comme lors du premier projet, étant donné que nous ne connaissons pas d'environnement de développement permettant de développer ensemble un même projet, et parce que nous préférons travailler ainsi, la majeure partie du projet s'est faite à deux, du moins au début. On a également préféré se regrouper pour faire les méthodes et classes compliquées (par exemple la méthode "valideTour" qui fait plus de 50 lignes et que nous n'avons pas réussi à compacter plus que ça) . Etant donné que les méthodes évoluent au fur et à mesure du code on peut aussi considérer que la javadoc a été rédigée à deux.

B. Contributions de Valentin

J'ai commencé par créer la classes "case" ainsi que ses dépendances "caseSimple" et "portail" pour pouvoir placer les bushi. Ensuite, j'ai créé la classe plateau permettant de créer un tableau de case et de gérer tout les déplacements et la suppression des bushis sur les cases, ce qui

permet de pouvoir créer un affichage assez simple avec une double boucle for. Une fois les bases de plateau et case finies, j'ai pu créer la classe bushi et ainsi placer ces bushi sur les cases du plateau. J'ai développé la classe partie pour permettre de gérer le jeu, de gérer les joueurs réalisé en majeur partie par Alexis, c'est cette classe qui définit qui doit jouer et quelles sont les conditions pour qu'il effectue telle ou telle action. Cette classe m'a aussi servi à créer une sauvegarde automatique dans le dossier save. Cette sauvegarde peut être chargée dans la classe gestionPartie, l'utilisateur peut choisir sa sauvegarde parmi toute celles créées. Celle-ci sont supprimé une fois la partie terminé ou dès que le joueur décide de la supprimer dans le menu principal. J'ai aussi effectué la gestion des exception et la vérification des données entrées par le joueur via la classe scanner. Enfin j'ai créé les méthodes permettant d'appliquer les règles, c'est à dire permettant de vérifier si un saut est possible puis l'effectuer, vérifier qu'un glissement est possible en fonction des données rentré par l'utilisateur (direction, distance) puis effectuer ce glissement. Tout ceci est développé dans la classe bushi. Pour finir, nous avons un peu modifié le diagramme de classe par rapport à l'original, j'ai donc régénérer ce diagramme en fonction du code.

C. Contribution d'Alexis

J'ai d'abord fait toutes les saisies utilisateurs (choisirDirection(), choisirDistance() …). J'ai aussi eu l'idée de créer la classe gestionPartie, ainsi j'ai pu y intégrer le menu principal avec les méthodes pour créer une nouvelle partie, créer un joueur, choisir qui débute, afficher les règles.. Valentin s'est occupé de toute la partie gestion des sauvegardes. Pour faire toutes ces saisies j'ai du refaire toutes les énumérations et y ajouter des méthodes getter pour transformer l'input de l'utilisateur (une lettre, pour que ce soit plus ergonomique et intuitif que de rentrer un chiffre) en un objet de l'énumération (c'est la méthode la plus compacte que j'ai trouvé pour gérer la saisie utilisateur sans faire des dizaines de "if" ou "switch case"). J'ai également redéfini tous les toString des classes pour faciliter l'affichage lors du débogage.

J’ai également eu l’idée de personnaliser l’affichage de fin de partie pour le rendre moins “triste” qu’un simple affichage du joueur gagnant. A la fin de la partie il y a désormais toute une série de statistiques sur la partie qui vient de se jouer (plus de détails dans la partie “Mode d’emploi illustré”). Nous avions aussi fait l’erreur de créer toutes les classes avec les attributs publics, alors au bout d’un moment j’ai du repasser tous les attributs en private pour respecter les bonnes pratiques, donc j’ai refais tous les setters et getter nécessaire puis je les aient intégré au code. Je me suis occupé de gérer toute la nomenclature du code, j’ai essaye de rendre le code le plus lisible possible en supprimant tout ce qui est inutile, j’ai renommé toutes les variables de la même manière : chaque nom de variable comme par son type puis est suivie par son nom commençant par une majuscule (exemple : Variable de type Bushi, son nom est singe, cela donne bushiSinge). Tous les paramètres de fonction respectent cette règle et ont un “ P “ à la fin de leur nom. Finalement je me suis occupé de la majeure partie de la rédaction du dossier.

IV. Bilan qualitatif du travail

A. Difficultées rencontrées

La plus grande difficulté, comme lors du premier projet, a été de s’organiser pour les phases où on ne pouvait pas travailler ensemble, car de base nous préférons programmer sur le même ordinateur et avancer ensemble. On a eu aussi beaucoup de difficulté pour comprendre le diagramme de classe de base et interpréter la fonction de chaque méthode, c’est pourquoi notre diagramme de classe final comporte pas mal de différences. De plus nous avons eu quelques difficultés au tout début du projet pour se familiariser avec la programmation objet et essayer de faire un lien entre notre ancien projet en C et le nouveau projet en Java, on ne voyait pas vraiment où placer les méthodes de vérification de tour, de saisie... Une autre difficulté a été de gérer notre temps, avec plus de temps nous aurions pu améliorer davantage notre programme et ajouter plusieurs fonctionnalités auxquelles nous avons pensé.

B. Améliorations possibles

Voici quelques idées d'amélioration possibles auxquelles nous avons pensé et que nous n'avons pas pu implanter, faute de temps.

- Intégrer du JavaFX pour avoir le jeu en interface graphique et jouable à la souris, cela permet non seulement une meilleure ergonomie mais aussi l'usage de couleurs sur le plateau plutôt que d'utiliser des glyphes différents pour une meilleure lisibilité
- Garder les joueurs en mémoire pour éviter de les recréer à chaque fois (par exemple lors d'une nouvelle partie, on demande soit de sélectionner un joueur soit d'en créer un nouveau)
- Essayer de réduire les quelques fonctions trop longues que nous avons écrit pour une meilleure lisibilité et faciliter le débogage
- Lors du choix des directions, supprimer les directions impossibles à effectuer, pareil pour le choix " glisse" et "saute", pour supprimer totalement le message " déplacement impossible" et demander directement les saisies en cas d'erreur

V. Mode d'emploi illustré

Les règles sont les même que les règles classiques du shing shang complétées par celles évoquées dans nos choix de conception.
Voici un scénario typique de notre programme.

- On arrive sur le menu principal demandant ce que l'on souhaite faire

Une saisie de l'utilisateur est nécessaire, il faut rentrer une des lettres indiquées à gauche.
Nouvelle partie permet de lancer une toute nouvelle partie
Charger partie permet de reprendre une partie déjà commencée

Supprimer partie permet de supprimer une sauvegarde
Règles du jeu permet d'afficher les règles du jeu
Quitter permet de quitter le jeu

- Supposons que l'on souhaite créer une nouvelle partie
- On entre la lettre n (majuscule ou minuscule peu importe)
- Une ligne nous indique qu'il faut saisir le nom du premier joueur
- Il faut ensuite choisir la couleur de son équipe
- Puis choisir le nom du second joueur
- La couleur du second joueur est choisie automatiquement en fonction de la saisie du premier joueur

```
Nom du joueur :
( Moins de 15 caractères )
Alexis

Choix de la couleur :
( N. Noir R. Rouge )
R

Nom du joueur :
( Moins de 15 caractères )
Valentin
```

- Après la saisie des joueurs un menu demande quel joueur doit commencer la partie, il est possible de choisir un joueur ou bien de choisir au hasard un des deux joueurs

```
Quel joueur doit commencer ?
1) Alexis
2) Valentin
3) Au hasard
```

- Le plateau s'affiche et la partie débute, en majuscule les Bushis rouges, en minuscule les pions noirs

```
  Y 0  1  2  3  4  5  6  7  8  9
X                Alex
0     D  L  S  .  .  S  L  D
1     L  S  .  P  P  .  S  L
2     S  .  .  .  .  .  .  S
3     .  .  .  .  .  .  .  .
4  .  .  .  .  .  .  .  .  .  .
5  .  .  .  .  .  .  .  .  .  .
6     .  .  .  .  .  .  .  .
7     s  .  .  .  .  .  .  s
8     l  s  .  p  p  .  s  l
9     d  l  s  .  .  s  l  d
                Vale
```

- Il faut alors saisir les coordonnées du pion à déplacer
- Il faut ensuite choisir l'action que l'on souhaite effectuer

Fin termine le tour

Glisse déplace le bushi si il n'y a aucun autre bushi entre la case de départ et celle d'arrivée
Saute permet de sauter par dessus un bushi sur une case adjacente
Annule permet de ressaisir les coordonnées du bushi

- Il faut ensuite choisir la direction du glissement/saut

(Au lieu des points cardinaux à l'origine nous avons préféré mettre des directions Droite, Gauche, Haut, Bas, trouvant cela plus intuitif)

- Si l'action est "glisse" il faut ensuite choisir la distance du déplacement

```
à Alexis de jouer !

Coordonnées du Bushi à déplacer :
x: 1
y: 1
Alexis : Choisissez une action

g) Glisse
s) Saute
a) Annule

g
g
Action : Glisse
Alexis : Choisissez la direction

H  - en Haut
HD - en Haut à Droite
D  - à  Droite
BD - en Bas à Droite
B  - en Bas
BG - en Bas à Gauche
G  - à  Gauche
HG - en Haut à Gauche
bd
Alexis : Veuillez saisir une distance :
1
```

- Les tours s'enchaînent selon les règles du jeu
- Une fois la partie terminée, un message de fin et des statistiques s'affichent puis il y a un retour au menu principal

```
zad remporte la partie !

--- Statistiques de la partie ---
Nombre de tours      : 15
Nombre de sauts      : 10
Nombre de ShingShang : 0

--- Statistiques de zad ---
Nombre de tours      : 8
Nombre de sauts      : 8
Nombre de ShingShang : 0

--- Statistiques de zad ---
Nombre de tours      : 7
Nombre de sauts      : 2
Nombre de ShingShang : 0
```

V. Ressentis personnels

A. Conclusion Valentin

Ce projet m'a permis de développer mes compétences en programmation objet avec laquelle j'avais un peu de mal à comprendre certains concept, ainsi que mes compétences en java. Cela m'a aussi permis de me familiariser avec l'IDE Eclipse. Enfin j'ai pu apprendre à travailler en groupe, à répartir les tâches et à rendre un code clair pour que Alexis puissent le comprendre sans problème pour développer ses méthode et pour que celui ci soit lisible si une autre personne veut faire évoluer notre programme ou se resservir de certaines classes.

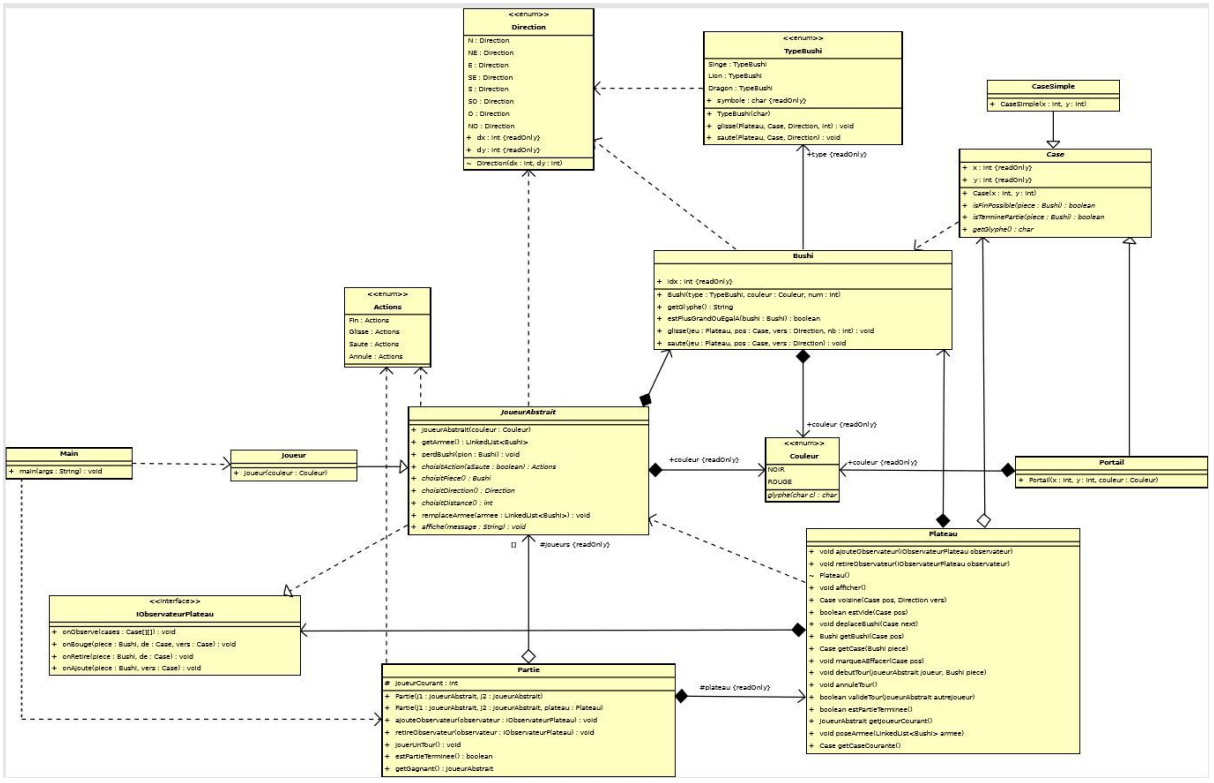
B. Conclusion Alexis

Ce projet m'a permis d'affiner mes connaissances en programmation orientée objet, en plus des notions déjà vues en cours. Comme lors du premier projet en C il m'a permit de me familiariser avec la programmation en groupe et de me rendre compte de la difficulté de répartir des tâches tout en travaillant ensemble. Il m'a aussi incité à faire preuve de rigueur dans mon code pour que l'autre soit apte à le reprendre et à comprendre directement à quoi il sert.

VI. Annexes

Note : les annexes sont aussi accessibles dans le dossier .zip rendu

Annexe 1 :



Annexe 2 : Notre diagramme de classe