Algorithme et Programmation

Mr Jack Pocket

Projet Java

DION François
GARDARIN Wilhem
Pauline PHILIPPE

A1 – Groupe 11

SOMMAIRE

Introduction

Présentation du projet

Dans le cadre de notre A1 à l'ISEP, il nous a été demandé de développer en langage Java le jeu Mr Jack Pocket. Le but principal de ce projet est d'apprendre la programmation orientée objet à travers le développement du jeu.

L'objectif final du projet consiste à créer un programme, pouvant être visualisé graphiquement à l'aide d'une interface graphique ici : Swing. Nous pouvions également créer une intelligence artificielle pour pouvoir jouer à un seul joueur cependant nous n'avons pas eu le temps de la concevoir.

Présentation du jeu

Mr Jack Pocket est un jeu de bluff et de déduction asymétrique pour 2 joueurs. L'un des joueurs incarne l'équipe de Sherlock Holmes et l'autre Jack l'Eventreur. L'objectif de Sherlock et sa bande est de déterminer l'identité de Mr Jack, tandis que l'objectif de Mr Jack est de ne pas se faire démasquer.

Présentation du principe

Chaque tour se joue en deux étapes : La Traque et l'Appel à Témoin. Durant la première étape, quatre actions sont réalisées, soit deux par joueur. Ces actions permettent par exemple de déplacer les Détectives autour du district, de modifier l'orientation ou la position des quartiers ou encore d'obtenir des cartes Alibi. Durant la deuxième étape, on procède à l'Appel à Témoin. Mr. Jack indique à l'Enquêteur si son personnage est ou n'est pas dans la ligne de vue des Détectives. L'Enquêteur pourra alors peut-être innocenter des personnages en retournant des quartiers sur leurs faces vides, diminuant ainsi le nombre de suspects pour resserrer l'étau autour de Mr. Jack.

Architecture et structure du projet

Pour structurer notre projet, nous avons fait appel à un certain nombre de classes, énumérations et interfaces, qui interagissent entre elles. Dans l'ordre des choses, nous avons d'abord créé une classe Position. En effet, que ça soit pour les alibis ou les détectives, chacun d'entre eux possède une position. Une position comprend quatre attributs : une ligne, une colonne, un sens (orientation NORTH, SOUTH, EAST, WEST) et un état (INGAME, RETURNED). Comme les détectives n'ont ni sens, ni état, nous avons défini par défaut leur sens comme "AUCUN" et leur état comme "NONE".

Par la suite, nous avons créé les enums AlibiName et Detective. Faire des enums plutôt que des classes était bien plus simple pour représenter les différents alibis et détectives. Par exemple pour les alibis, chacun possède un nombre de sabliers différents, mais fixes. Ainsi il était bien plus logique de faire un énum regroupant les alibis, en lui donnant comme paramètre directement le nombre de sablier qui lui correspond.

Pour ce qui est de l'enum AlibiName, il faut savoir que dans le jeu, les alibis situés en haut à droite, en haut à gauche, et au centre en bas doivent avoir une orientation spécifique en début de jeu. En effet, lorsqu'on initialise une partie, chaque détective est face à un mur.

La méthode "placerAlibi()" se charge donc de mélanger une liste de neufs alibis, et pour les trois qui ont une position proche d'un détective, leur orientation est définie en conséquence.

Pour ce qui est de l'enum Detective, chaque détective est placé en début de partie à un emplacement précis (voir règle du jeu). Ainsi, la méthode "placerDetective()" se charge de placer chaque détective à leur place au début de la partie, tout en initialisant leur sens à "AUCUN" et leur état à "NONE".

Ensuite, nous avons créé les différents jetons (x7) jouables par les utilisateurs. Tout comme dans le TP du jeu d'échecs, nous avons créé une Superclasse Jeton qui possède 2 méthodes abstraites : "action()" et "getName". Comme indiqués par leurs noms, la méthode action va définir ce que fait un jeton en particulier, et la méthode getName va simplement faire un print du nom du jeton afin d'afficher à l'utilisateur quel jeton lui est disponible. Cette Superclasse possède 7 classes filles qui représentent les 7 jetons. Voici les différents jetons et leur fonctionnement (en console, car notre jeu n'a été développé que pour un print console).

- Jeton3Personnages : Prend en compte la liste des Détectives et le déplacement possible associé à l'action. Demande à l'utilisateur de saisir le nom du détéctive à déplacer ("TOBY, HOLMES ou WATSON"). Demande ensuite à l'utilisateur de saisir "0" ou "1" qui correspond à l'avancement du détective dans le sens horaire. Enfin, nous avons défini une méthode "move1of3Characters()" dans l'enum Détective qui prend en compte le mouvement choisi et la position actuelle du détective pour lui attribuer sa nouvelle position.

- JetonAlibi : Prend en compte la liste d'alibi et la mélange (le mélange est différent de celui fait en début de partie lors de l'initialisation des alibis sur le plateau, cf. règle du jeu).

Une fois cette liste mélangée, on tire un alibi de cette liste. Si cet alibi est différent de l'identité de MrJack alors deux cas de figure sont possibles : si c'est MrJack qui a tiré cette carte, il reçoit le nombre de sablier indiquée sur la carte. Si c'est le détective, la carte de l'alibi tirée est retournée sur le plateau et n'est plus en jeu. Si on tire une carte identique à celle de l'identité de MrJack, alors cette carte n'est pas dévoilée, on en tire simplement une deuxième, différente de la précédente car on ne remélange pas la liste entre les deux tirages, on choisit simplement un alibi à un index différent dans la liste. En ce qui concerne les méthodes utilisées par cette classe : Pour tourner une tuile, on utilise la méthode "tournerTuile()" qui a été définie dans l'enum AlibiName. Cette méthode est simple : on crée une variable de type Position qui va reprendre la position de la carte Alibi que l'on a pioché, et va "set" l'état de la tuile à RETURNED. Cependant il y a un cas particulier que nous avons eu à traiter. En effet la carte de l'alibi "LANE" possède un sens en croix lorsqu'elle est retournée, c'est-à-dire qu'elle ne possède plus vraiment de sens puisqu'elle est visible de n'importe quel côté. Ainsi, on "set" l'état de la tuile sur "AUCUN", en plus de la retourner.

- JetonDeplacerHOLMES(ou TOBY, ou WATSON) ⇒ 3 Classes "identiques" : Etant donné que ces trois classes sont identiques et que le seul élément changeant est le nom du détective que l'on souhaite déplacer, il n'y a pas besoin de présenter ces classes trois fois.

Ainsi les explications qui vont suivre sont communes aux 3 classes. Prenons par exemple le détective TOBY. On va dans un premier temps définir une liste d'int qui prend deux éléments "1" et "2". Cette liste va définir le déplacement possible de TOBY (Bien sûr, comme pour tous les inputs des utilisateurs, on s'assure qu'ils ne puissent pas écrire un autre déplacement que ceux qui leurs sont proposés avec un do-while). Le mouvement du détective est ensuite assurée par la méthode "move1Character()" qui est définie dans l'enum Détective (même principe que pour la méthode move1of3Characters).

- JetonEchangeTuile: Le jeton échange tuile fonctionne de la manière suivante : On crée une liste qui reprend chacun des alibis (x9 éléments), à l'aide de deux boucles do-while, on demande à l'utilisateur de saisir la tuile qu'il souhaite déplacer, puis celle avec laquelle il souhaite faire le changement. Ensuite, grâce à la méthode "echangerPositionAlibi()" qui prend en paramètre les 2 alibis que l'on a saisi, ces 2 alibis échangent leurs places.

- JetonRotationTuile: Le jeton rotation tuile fonctionne de la même manière que le jetonEchangeTuile, sauf que dans un second temps on ne demande pas à l'utilisateur de saisir un deuxième alibi, mais cette fois un sens (NORTH,SOUTH,EAST,WEST). Ensuite, la rotation de la tuile est assurée par la méthode "rotationTuile()" présente dans la classe AlibiName. Rappelons qu'il y a 8 jetons à jouer dans MrJack, celui-ci étant présent en double dans le jeu (jouable deux fois).

- Classe PlayerMrJack : Cette classe comprend plusieurs méthodes.

setMrJackCard() ⇒ Création d'un ArrayDeque avec comme éléments tous les alibis de notre liste d'alibis. On tire la première carte (.pollfirst), c'est l'identité de MrJack.

getMrJackCard() ⇒ Méthode qui retourne son identité.

sablierAddFinTour() ⇒ Ajoute un sablier à MrJack (si il n'est pas visible en fin de tour).

sablierAddAlibi ⇒ si MrJack joue le jetonAlibi, le nombre de sablier présent sur la carte qu'il a tiré lui est attribué.

getSablier ⇒ Méthode qui retourne le nombre de sablier au total.

- Enum PlateauJeu (Enum principale) : L'enum PlateauJeu est la plus grand énum du jeu. Dans un premier temps, on crée à l'intérieur de cette énum une classe PlateauMrJack.

Le principe de cette classe est simple : Les alibis et les détectives ont une position représentée par une ligne et une colonne (pour ce qui nous intéresse ici). Nous allons placer dans un tableau, en fonction de leur ligne et leur colonne les alibis puis dans un second tableau, les détectives. Comme ces deux classes sont distinctes alors elles ne sont pas du même type et il n'est donc pas possible de les mettre toutes les deux dans un même tableau. Ainsi on crée une interface "PositionableObject" que les enums AlibiName et Détective implémentent pour ainsi créer une méthode getExtendedBoard(), qui va renvoyer un tableau 5x5 de type PositionableObject, avec dedans, les positions des Alibis et des Détectives. Ainsi on a regroupé 2 types dans un même tableau. Notre "board" est donc créé.

La suite de l'enum concerne le déroulement du jeu. Dans la méthode PlateauJeu, on va dans un premier temps placer les Alibis et les Détectives et attribuer à MrJack une identité. Ensuite on va ajouter à une liste nos jetons, que nous aurons créés les lignes d'avant. Ensuite, en accord avec les règles du jeu, la partie va se dérouler. D'un côté nous avons les tours pairs. Lors du tour pair, le détective joue le 1 er et le dernier jeton. MrJack lui, joue le 2ème et le 3ème jeton. Lors du tour impair, c'est l'inverse. A la fin de chaque tour, on vérifie si la partie se termine ou non (si MrJack possède ou non 6 sabliers ou +, si le compteur de tour est supérieur à 8). On vérifie bien sûr également si MrJack est visible ou non à la fin de chaque tour. S'il est dans la ligne de mire des détectives, alors on retourne toutes les cartes qui ne sont pas visibles par les détectives et inversement. Le programme gère également les exceptions. Si par exemple MrJack possède un nombre strictement supérieur à 5 sabliers et qu'en même temps, il ne reste plus que l'alibi de MrJack en jeu, alors le détective l'emporte si MrJack est visible (présent dans la liste des alibis visibles). En revanche, si MrJack ne figure pas dans la liste des Alibis visible, alors c'est lui qui l'emporte. A chaque fin de tour pair, il reste 4 jetons dans la liste des jetons qui sont alors proposés au tour impair. A chaque fin de tour impair, on reprend notre liste de jetons vide à laquelle on rajoute à nouveau les 8 jetons que l'on mélange à nouveau grâce à la méthode "jetonShuffle()" présente dans l'enum PlateauJeu. Pour gérer l'affichage en console du board, on a fait une méthode "updateBoard()" qui s'occupe de mettre à jour le board en fonction des jetons joués. La méthode "printBoard()" print le tableau en console.

Enfin, comment est gérée la visibilité des alibis par les détectives ? Nous avons créé dans la classe Detective une méthode "isVisible()" qui, avec des boucles if, testent toutes les positions possibles des détectives. Nous avons défini la position des lignes et des colonnes comme ceci :

L/C	0	1	2	3	4	C/L
0		Détective	Détective	Détective		0
1	Détective	Alibi 1	Alibi 2	Alibi 3	Détective	1
2	Détective	Alibi 4	Alibi 5	Alibi 6	Détective	2
3	Détective	Alibi 7	Alibi 8	Alibi 9	Détective	3
4		Détective	Détective	Détective		4
L/C	0	1	2	3	4	C/L

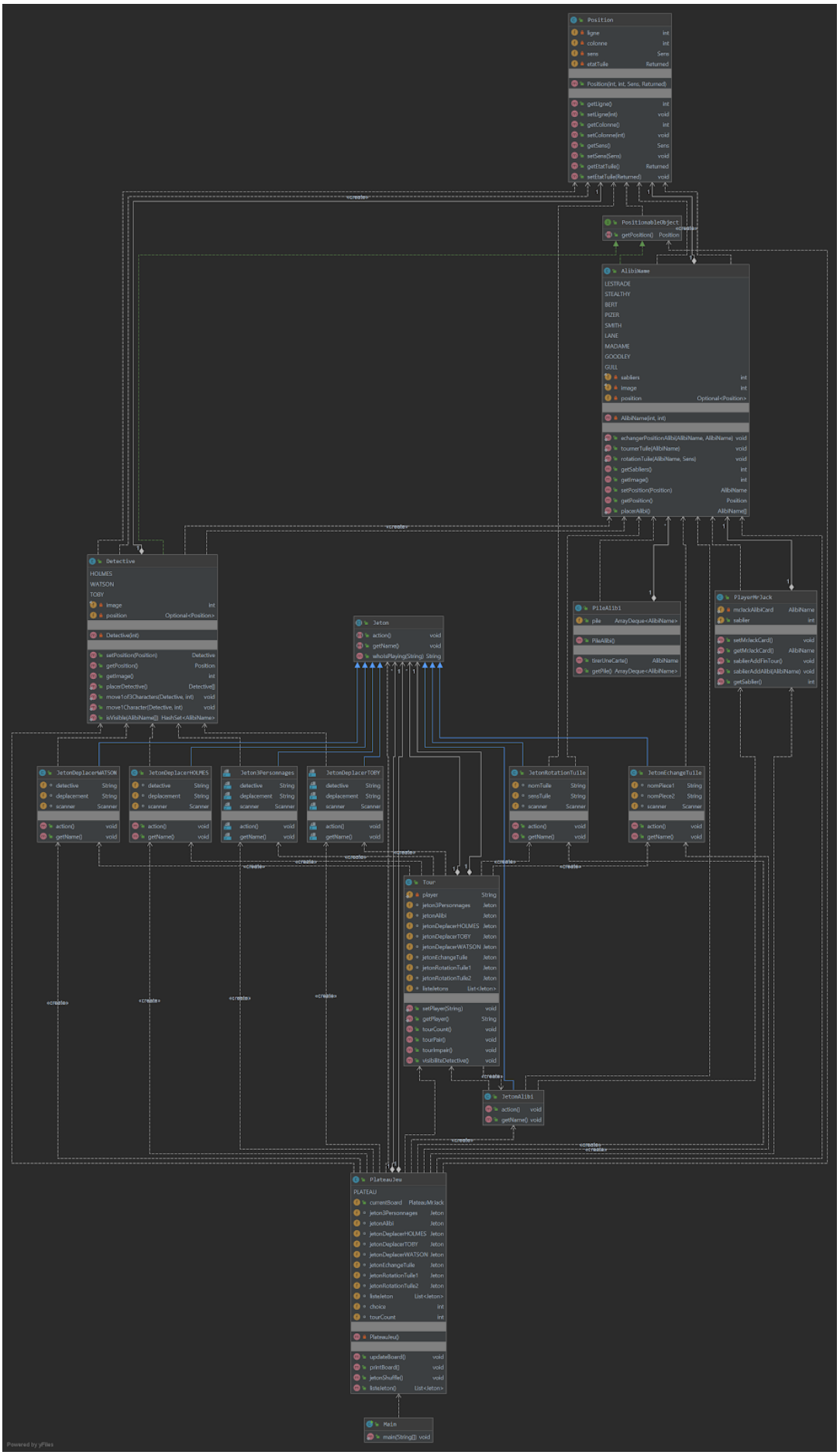
Ainsi, si le détective se trouve en (L, C) ⇒ (0,1) alors il est face aux alibis 1,4,7. Pour le sens, nous avons décidé qu'une carte était orientée ("NORTH") lorsque le bout de son T pointe au nord. En début de partie les détectives ont comme position :

HOLMES ⇒ (1,0)
WATSON ⇒ (1,4)
TOBY ⇒ (4,2)

Alors la position d'Alibi 1 est initialisée au début à l'est, Alibi 8 au nord et Alibi 3 à l'ouest. Pour déterminer si un alibi est visible ou non, voici comment on a procédé avec des boucles if :

Si un des détectives a comme position (L ⇒ 0,..,4, C ⇒ 0,...4) (<== ensemble des positions possibles sauf L/C = 0,0 et 0,4 et 4,0 et 4,4) alors par exemple pour notre détective situé en (0,1), si le sens d'Alibi 1 est différent de SOUTH, il est visible (première boucle if). Maintenant, si le sens d'Alibi 1 est EAST ou WEST ou AUCUN (cas LANE retournée) et que le sens d'alibi 4 est différent de SOUTH, alors Alibi 4 est aussi visible (deuxième boucle if). Si le sens d'Alibi 4 est EAST ou WEST ou AUCUN et que le sens d'alibi 7 est différent de SOUTH, alors Alibi 7 est aussi visible (troisième boucle if), à chaque boucle if, on ajoute l'alibi qui correspond à notre liste des visibles. Cette liste des alibis visibles est un HashSet, très utile pour éviter qu'un alibi soit en double dans notre liste (cas où deux détectives voient un même alibi). A la fin, la méthode nous renvoie notre liste d'alibis visibles, et c'est dans "PlateauJeu()" qu'est dit si ces alibis vont être retournés, ou non à la fin d'un tour, (si MrJack figure dans la liste ou non).

Voici maintenant un diagramme UML du projet : (très peu visible à moins de faire un gros zoom ou d'avoir un télescope, mais la photo a été ajoutée aux fichiers du projet ainsi que sur Github). Toutefois, la classe PileAlibi ne figure plus dans le projet.


Position
ligne int
colonne int
sens Sens
etatTuile Returned
Position(int, int, Sens, Returned)
getLigne() int
setLigne(int) void
getColonne() int
setColonne(int) void
getSens() Sens
setSens(Sens) void
getEtatTuile() Returned
setEtatTuile(Returned) void
PositionableObject
getPosition() Position
AlibiName
LESTRADE
STEALTHY
BERT
PIZER
SMITH
LANE
MADAME
GOODLEY
GIUL
sabliers int
image int
position Optional<Position>
AlibiName(int, int)
echangerPositionAlibi(AlibiName, AlibiName) void
tournerTuile(AlibiName) void
rotationTuile(AlibiName, Sens) void
getSabliers() int
getImage() int
setPosition(Position) AlibiName
getPosition() Position
placerAlibi() AlibiName[]
Detective
HOLMES
WATSON
TOBY
image int
position Optional<Position>
Detective(int)
setPosition(Position) Detective
getPosition() Position
getImage() int
placerDetective() Detective[]
move1of3Characters(Detective, int) void
move1Character(Detective, int) void
isVisible(AlibiName[]) HashSet<AlibiName>
Jeton
action() void
getName() void
whoIsPlaying(String) String
PileAlibi
pile ArrayDeque<AlibiName>
PileAlibi()
tirerUneCarte() AlibiName
getPile() ArrayDeque<AlibiName>
PlayerMrJack
mrJackAlibiCard AlibiName
sablier int
setMrJackCard() void
getMrJackCard() AlibiName
sablierAddFinTour() void
sablierAddAlibi(AlibiName) void
getSablier() int
JetonDeplacerWATSON
detective String
deplacement String
scanner Scanner
action() void
getName() void
JetonDeplacerHOLMES
detective String
deplacement String
scanner Scanner
action() void
getName() void
JetonPersonnages
detective String
deplacement String
scanner Scanner
action() void
getName() void
JetonDeplacerTOBY
detective String
deplacement String
scanner Scanner
action() void
getName() void
JetonRotationTuile
nomTuile String
sensTuile String
scanner Scanner
action() void
getName() void
JetonEchangeTuile
nomPiece1 String
nomPiece2 String
scanner Scanner
action() void
getName() void
Tour
player String
jetonPersonnages Jeton
jetonAlibi Jeton
jetonDeplacerHOLMES Jeton
jetonDeplacerTOBY Jeton
jetonDeplacerWATSON Jeton
jetonEchangeTuile Jeton
jetonRotationTuile1 Jeton
jetonRotationTuile2 Jeton
listeJetons List<Jeton>
setPlayer(String) void
getPlayer() String
tourCount() void
tourPair() void
tourImpair() void
visibiliteDetective() void
JetonAlibi
action() void
getName() void
PlateauJeu
PLATEAU
currentBoard PlateauMrJack
jetonPersonnages Jeton
jetonAlibi Jeton
jetonDeplacerHOLMES Jeton
jetonDeplacerTOBY Jeton
jetonDeplacerWATSON Jeton
jetonEchangeTuile Jeton
jetonRotationTuile1 Jeton
jetonRotationTuile2 Jeton
listeJeton List<Jeton>
choice int
tourCount int
PlateauJeu()
updateBoard() void
printBoard() void
jetonShuffle() void
listeJeton() List<Jeton>
Main
main(String[]) void
«create»

Mode d'emploi du jeu

Dans un premier temps, il faut ouvrir le projet avec un IDE Java.

Ensuite, il faut ouvrir la console sur une hauteur de ½ de la taille de l'écran. S'échanger l'ordinateur pour jouer et suivre les instructions en console.

Pour le joueur incarnant Mister Jack, l'identité de MrJack lui est montrée à chaque fois que c'est son tour.

Conclusion

La production de ce projet en Java nous a permis de renforcer et de mettre en pratique nos connaissances du langage. Ce projet représente pour nous, le premier jeu que nous codons en informatique et n'ayant pas eu le temps de programmer l'interface graphique, nous aimerions le faire prochainement. En effet, il est relativement frustrant pour nous de n'avoir "qu'une" version console qui ne représente pas le résultat visuel que nous attendions.

Problèmes rencontrés

Comme nous débutons tous les trois en Java, notre progression a été très différentes entre les différents membres du groupe, ce qui a entraîné une grande disparité du temps de travail de chacun pour l'avancement du projet.

Lien Githup : https://github.com/francois-71/MrJackPocketFPW