



Durand Faustine
Hamouma Mickael
Brunet-Manquat Ivan
Coquilhat Mattéo

Rapport de projet :

Jeu de dames en java

Licence 2 informatique
2015
Projet informatique
HLIN405

Encadrant:
Abdelhak-Djamel Seriai





REMERCIEMENTS

Nous tenons à exprimer notre gratitude et nos remerciements à toutes les personnes qui nous ont aidé de près ou de loin à la réalisation de notre projet.

Nous remercions tout d'abord Mr. Seriai notre encadrant pour son aide et ses encouragements tout au long du projet.

Nous remercions également Mr. Sylvain Daudé enseignant en informatique à la faculté des sciences pour son aide ponctuelle et ses conseils.

Nous tenons aussi à remercier M. Jimmy Benoit étudiant en master informatique pour sa disponibilité et son aide pour la partie graphique de notre projet.

SOMMAIRE

I. Introduction

II. Analyse et conception

- Analyse du travail à réaliser
- Structure du projet

III. Réalisation et implémentation

- Description des algorithmes de la mécanique de jeu
- Sérialisation
- Interface graphique
- Intelligence artificielle

IV. Problèmes rencontrés et solutions apportées

VI. Conclusion

VII. Annexe

I. Introduction

Le jeu de dames est un célèbre jeu de réflexion joué à deux, se composant d'un damier et d'une vingtaine de pions dont le but est de manger ou d'immobiliser toutes les pièces du joueur adverse. Même si le but du jeu de dames reste inchangé, il existe toutefois différentes manières d'y jouer, notamment avec les règles internationales, ces règles sont données sur ce [lien](#).

Chacun des membres de ce groupe, Ivan Matteo Faustine et Mickael, appréciant le jeu de dames mais ne le connaissant que sous sa forme classique du plateau de bois, et chacun trouvant intéressant, de le modéliser sous la forme d'une application Java nous avons naturellement choisi ce projet. Il nous a donc été demandé de programmer un jeu de dame en Java en se basant sur les règles internationales. Afin d'y parvenir, il nous a fallu analyser tout d'abord le sujet à l'aide de diagrammes UML de cas d'utilisation et de classes et nous organiser en nous répartissant les tâches à accomplir, du jeu sur console au jeu graphique voir partie *analyse et conception*, puis nous sommes passés à l'implémentation du programme à proprement parler, la description des algorithmes se trouve donc dans *Réalisation et implémentation*, enfin nous avons bien évidemment rencontrés beaucoup d'obstacle qui seront listés dans *Problèmes rencontrés et solutions apportées*.

II. Analyse et conception

Analyse du travail à réaliser :

Un des objectifs principaux de ce projet a été de nous habituer au travail en groupe et aux notions de projet. Organiser son temps, établir les tâches sous la forme d'un diagramme de Gantt ou encore nous obliger à respecter notre cahier de charge. A la suite de nos premiers entretiens avec Mr. Seriai, l'enseignant en charge de la supervision de notre groupe, nous avons ainsi désigné Mickael comme chef de projet.

Nous avons ensuite et dans un premier temps travaillé ensemble sur les premières classes à établir (notre premier diagramme de classe en UML est donné en annexe anx1) afin d'établir un socle commun au sein de notre groupe en codant en dure (le joueur doit lire le code pour connaître les règles du jeu).

Voici notre planning prévisionnel :

| taches/Dates | 22/01/15 | 29/01/15 | 05/02/15 | 12/02/15 | 19/02/15 | 26/02/15 | 05/03/15 | 12/03/15 | 19/03/15 | 26/03/15 | 02/04/15 | 09/04/15 | 16/04/15 | 26/04/15 |
|--|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| Analyse du sujet | tous | | | | | | | | | | | | | |
| Etablissement des différents diagrammes | | tous | | | | | | | | | | | | |
| Rédaction des fonctions principales | | | Mattéo | et | Faustine | | | | | | | | | |
| Codage du début de la classe plateau | | | | Mickaël | | | | | | | | | | |
| Codage des autres classes | | | | Ivan et | Faustine | | | | | | | | | |
| codage des fonctions des rafles pour les pions | | | | Faustine | et | Mattéo | | | | | | | | |
| codage des fonctions des rafles pour les dames | | | | | | | | Faustine | et | Mattéo | | | | |
| Tests | | | | | | | tous | | | | | | | |
| codage de la partie graphique | | | | | | | Ivan | et | Mickaël | | | | | |
| codage des classes externes | | | | | | | Ivan | | | | | | | |
| Regroupement partie graphique/ Plateau | | | | | | | | | | | tous | | | |
| Mise en place d'un IA | | | | | | | | | | | tous | | | |
| Derniers tests | | | | | | | | | | | | tous | | |
| Rédaction rapport | | | | | | | | | | | | tous | | |

Notre but était de commencer à élaborer un programme sous forme console en déterminant les algorithmes qui nous donneraient les choix de déplacements des pions tout en respectant les règles internationales, en particulier la règle qui stipule que la prise du plus grand nombre de pions est obligatoire. Cette règle engendrant un algorithme assez compliqué où nos notions sur les fonctions récursives furent d'une grande utilité.



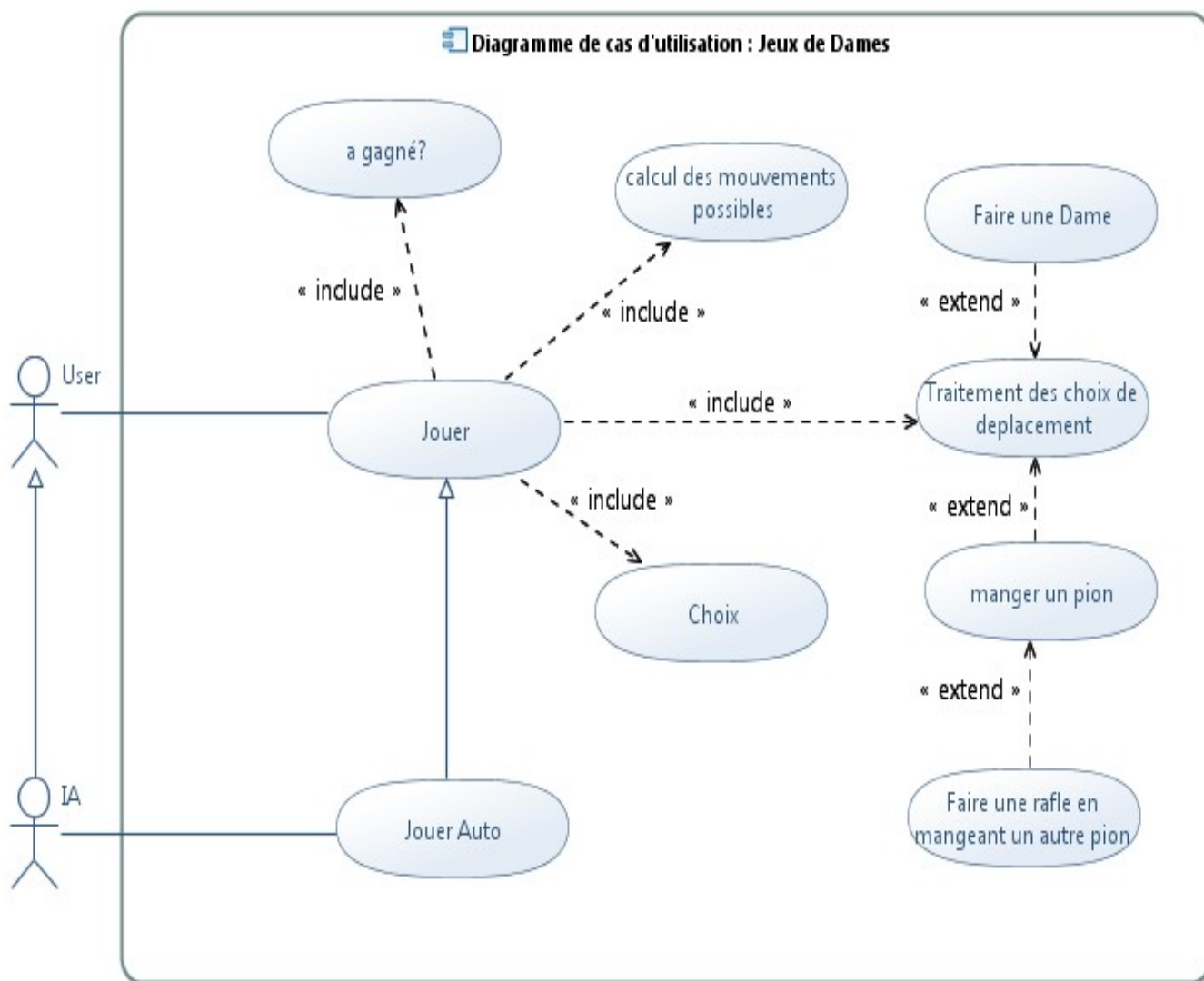
Il nous a également été demandé d'utiliser java comme langage informatique pour programmer ce jeu. Les avantages fournis par ce langage sont nombreux. La disparition des pointeurs nous a notamment simplifié la tâche et nous a permis de programmer plus vite sans rencontrer de problème.

Afin de programmer en java, nous avons d'abord écrit nos algorithmes sur SublimeText, puis intégré ces algorithmes sur eMacs et enfin nous les avons copiés sur eclipse que nous avons fini par adopter comme éditeur de base. Ce logiciel a été principalement utile pour les vérifications et les corrections effectuées automatiquement.

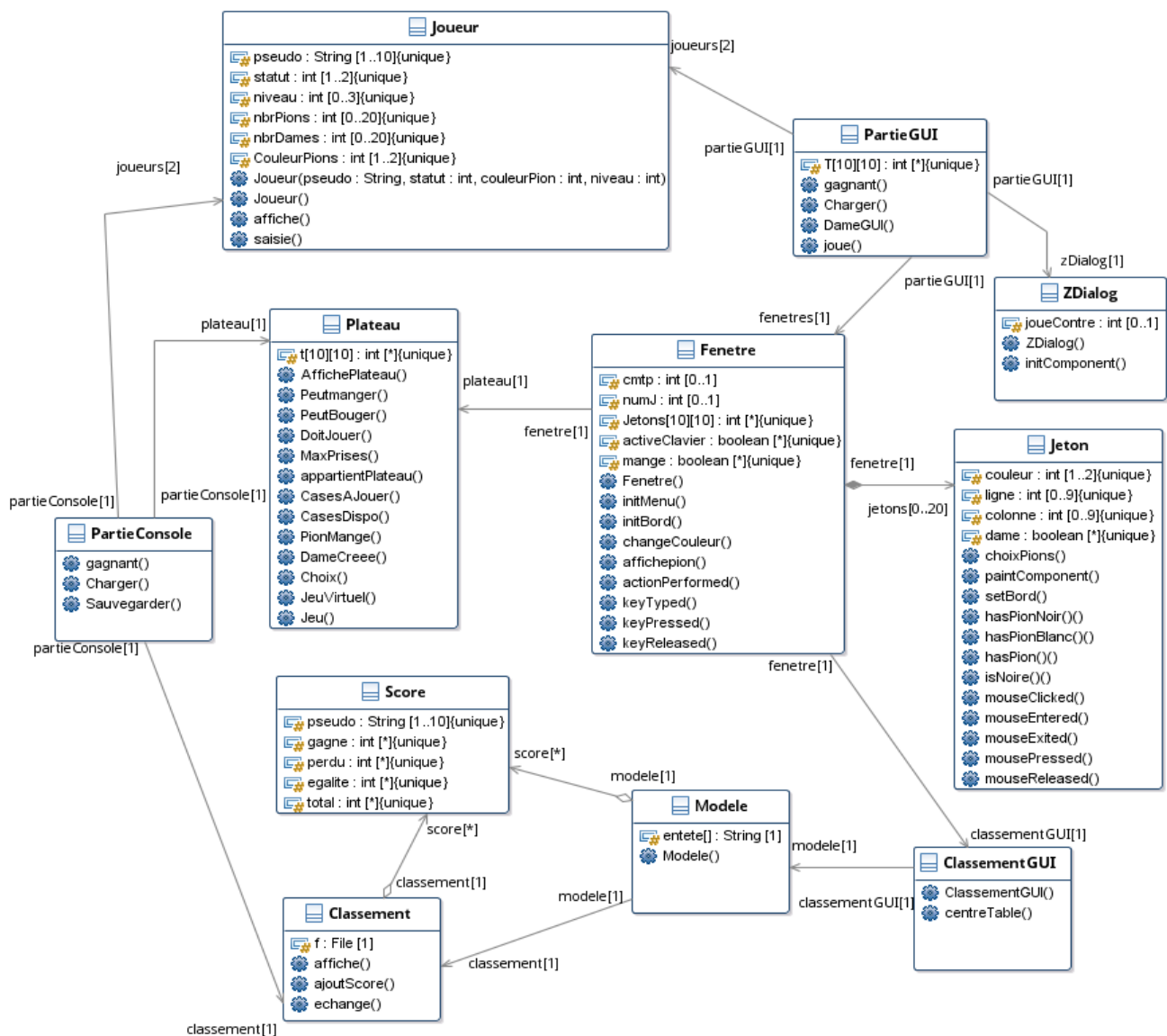
Par la suite il nous a fallu définir les étapes à accomplir afin de mener à bien ce projet. Il a donc été décidé d'implémenter premièrement une version console du jeu de dame, pour après programmer une interface graphique, un moyen de pouvoir quitter et sauvegarder une partie en cours de jeu, et enfin de créer une intelligence artificielle capable de jouer contre un joueur seul.

Après plusieurs premiers test et après avoir imaginé de nombreux cas de figures, nous sommes arrivé à une configuration du jeu à travers deux schéma :

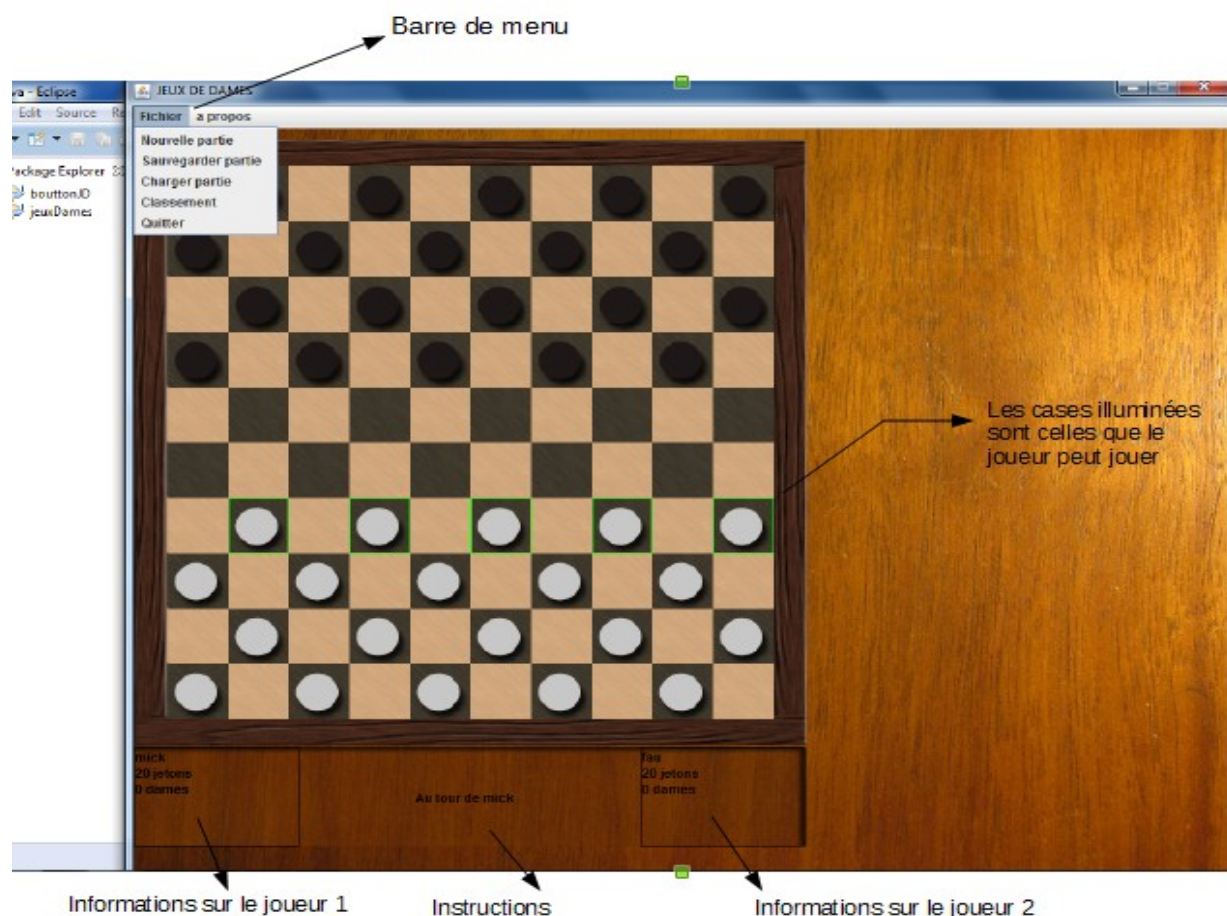
un schéma des cas d'utilisation :



et un diagramme de classe, structurant le projet:



Voici enfin comment nous avons décidé de modéliser notre projet:



Structure du projet :

Il y a deux classes principales qui permettent au jeu de se dérouler de différentes manières, (il y a donc deux fonctions main). La classe PartieConsole fait marcher le jeu sous console et la classe PartieGraphique dans une fenêtre.

Ci-suit une description des principales classes utilisées :

Plateau:

La classe indispensable à notre programme. C'est dans cette classe que vont être adaptées toutes les règles du jeu via de nombreuses fonctions.

Elle est composée d'un tableau qui représentera le damier avec pour valeurs:

- 1: jeton blanc
- 10: dame blanc
- 2: jeton noir
- 20: dame noir
- 0: case vide

Joueur:

-C'est la classe regroupant les informations d'un joueur à savoir son pseudonyme, le nombre de pions, le nombre de dames, son statut (réel ou virtuel), la couleur de ses pions et enfin le niveau du joueur dans le cas où il s'agirait d'un joueur virtuel. Cette classe comporte également des méthodes basiques de saisie et d'affichage. L'initialisation des joueurs en mode graphique se fait via des menus.
(annexe 2)

PartieConsole:

-C'est la classe qui va lancer la partie sous console.
-Elle est composée de deux joueurs.
-C'est ici que la fonction main va faire marcher le jeu sous console tant que la partie n'est pas terminée. Cette classe permet également de sauvegarder et de charger une partie.
(Anx.3)

Classement:

Cette classe permet de faire un classement des joueurs en fonction du nombre de parties gagnées, perdues, ou qui se termine par une égalité. Nous avons décidé de classer les joueurs tel que :

- Le joueur qui a gagné le plus de parties est premier.
- Si 2 joueurs ont le même nombre de parties gagnées, celui qui en a perdu le moins est premier.
- Si 2 joueurs ont le même nombre de parties gagnées et de parties perdues, celui qui a fait le plus d'égalités est premier.

Score:

Cette classe représente un score de Joueur à savoir son pseudo, son nombre de parties jouées, gagnées, perdues. Elle est utilisée par la classe classement qui contient une ArrayList de scores.

Fenetre:

La classe indispensable de l'interface graphique.

Elle hérite de JFrame et implémente les interfaces ActionListener et KeyListener. C'est donc ici aussi que les interactions entre la souris ou le clavier et notre interface graphique pourront avoir lieu.

Elle est ordonnée grâce à des JPanel et des Layout.

Jeton:

La classe représentant une case qui est soit vide blanche ou vide noire soit constituée d'un pion ou d'une dame (nous avons créé une classe héritière de JButton).

La case change si on change sa valeur principale «couleur»:

1:pion blanc

10:dame blanche
2:pion noir
20:dame noire
0:case noire vide
-2: case blanche vide

PartieGUI:

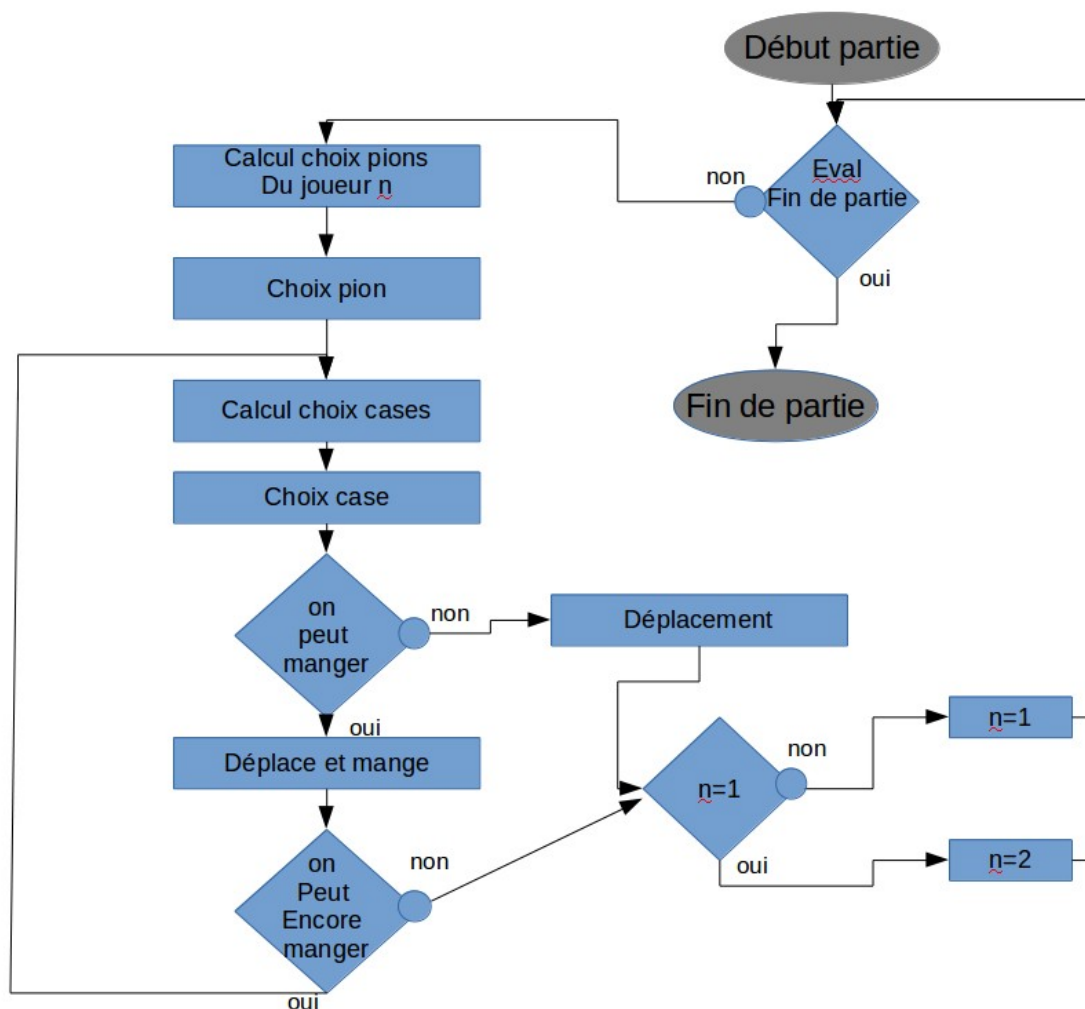
Cette classe a les mêmes fonctionnalités que la classe partieConsole mais sous forme graphique, donc dans une fenêtre.

(Anx.5)

III.Réalisation et implémentation :

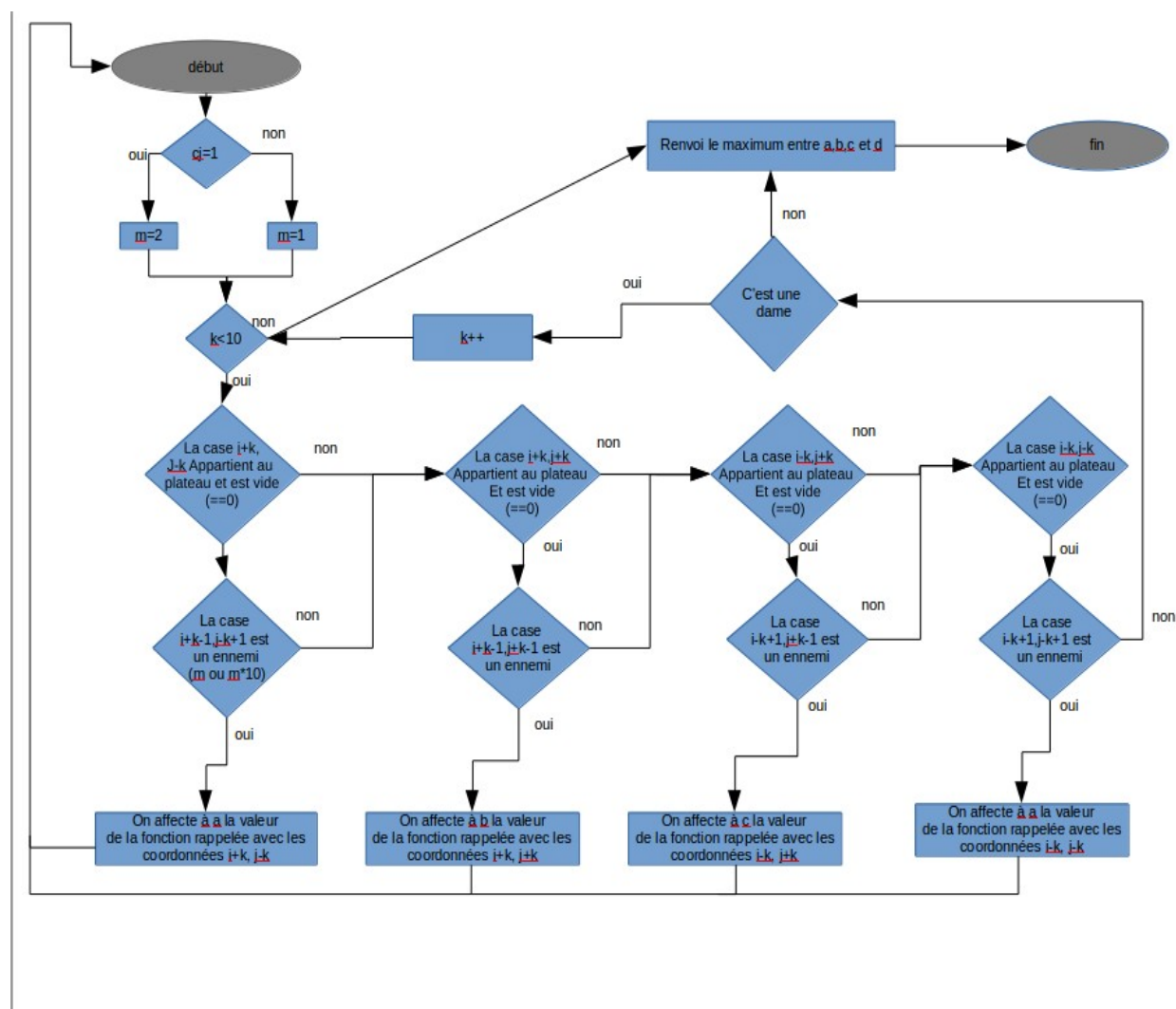
Descriptions des algorithmes de la mécanique de jeu :

Voici le déroulement d'une partie, utilisant diverses fonctions, que ce soit sous le mode console ou le mode graphique.



algorithmes principaux de la classe Plateau :MaxPrises:

Renvoie le maximum de prises à partir d'une case passée en paramètre, fonction récursive essentielle de la classe plateau. (voir annexe 6 pour l'algorithme)

Jeu:

Déroulement d'un tour d'un joueur.

DoitJouer:

Renvoie les pions que le joueur doit jouer au début de son tour.

CasesAJouer:

Renvoie les cases que le joueur doit jouer en fonction du pion choisit par le joueur en début de tour.

données: 3 entiers i, j: coordonnées de la case choisie et numéro du joueur courant cj.

résultat: max, vecteur de coordonnées des cases à que le joueur peut jouer à partir d'une pièce.

variables: a,b,c,d des entiers qui comptabilisent le nombre de pièces ennemis mangés selon les quatre directions possibles. nul vecteur de coordonnées empêchant qu'une pièce ne puisse manger deux fois un adversaire. s et s2 deux entiers qui permettront de savoir quel case permet le maximum de prises. o et n deux booléens permettant le contrôle des whiles. ja un entiers qui représente les pions ennemis (*10 si c'est une dame) et enfin un entier k, un compteur.

Début:

Si la pièce peut manger :

initialisation de ja en fonction du joueur courant.

(Cet algorithme est en deux partie, l'une pour les pions, basée sur des conditions intuitive, le détail de cette partie ne présentant pas d'intérêts nous n'explicitons ici que le détail de la seconde partie, pour les dames.) (...)

k=1

tant que la case courante appartient au plateau et n=true:

si o=true

si la case courante contient un ennemi

o=false et nul prend les coordonnées de l'ennemi.

Sinon si la case courante contient un pion allié

n=false

fin si

sinon s2 prend Maxprises de la case courante avec nul en paramètre.

Si s2>s

max est réinitialisée avec les coordonnées de la case courante

sinon

si il est égal

on ajouter les coordonnées de la case courante à max.

fin si

fin si

on reinitialise k, o, n, s et s2 et on réitère le while pour chaque diagonale autour de la dame.

renvoyer max

sinon **renvoyer** le résultat de CasesDispo, qui renvoi toutes les cases disponibles pour un pion ne pouvant pas manger.

pionMange:

Détermine si un pion a été mangé par le dernier déplacement effectué, il procède ensuite à l'élimination du pion.

PeutBouger:

Détermine à partir d'un pion s'il peut bouger (sans manger aucun pion).

PeutManger:

Détermine à partir d'un pion s'il peut manger un pion adverse.

AppartientPlateau:

Determine si une case donnée appartient toujours au plateau (utilisé pour des tests dans les while d'autres fonctions)

Choix:

Écoute le choix de pion/case du joueur en fonction de ceux qui ont été proposé

AfficheChoix:

Affiche les choix disponible pour le joueur courant.

affiche: Affiche le plateau (Uniquement pour la version console).

Serialisation/Désérialisation :

Nous avons pensé utile de permettre aux joueurs de sauvegarder une partie avant de quitter et de pouvoir la charger ultérieurement.

Nous avons alors fait face à un problème : Comment stocker des informations provenant du jeu, et être capable de les restaurer une fois le jeu relancé. Il nous était impossible de stocker ces informations (pseudo, nombre de pions, coordonnées des pions, etc..) dans des variables du jeu car une fois le jeu quitté, ces données étaient irrécupérables. Nous avons ainsi décidé d'utiliser un fichier texte pour stocker les données d'une partie.

Modèle de fichier de sauvegarde

```
0 2 0 2 0 2 0 2 0 2
2 0 2 0 2 0 2 0 2 0
0 2 0 2 0 2 0 2 0 2
2 0 2 0 2 0 2 0 2 0
0 0 0 0 0 0 0 0 0 0 // Tableau de la partie
0 0 0 0 0 0 0 0 0 0
0 1 0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0 1 0

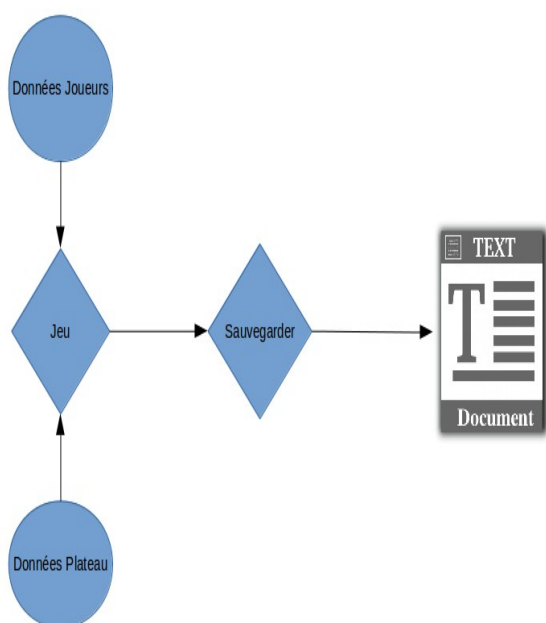
-----
pseudo
statut
NbrPions // Joueur 1
NbrDames
CouleurPions
NiveauIA
-----
pseudo
statut
NbrPions // Joueur 2
NbrDames
CouleurPions
NiveauIA
-----
cj // numero du joueur qui commencera à jouer
```

Exemple de fichier de sauvegarde

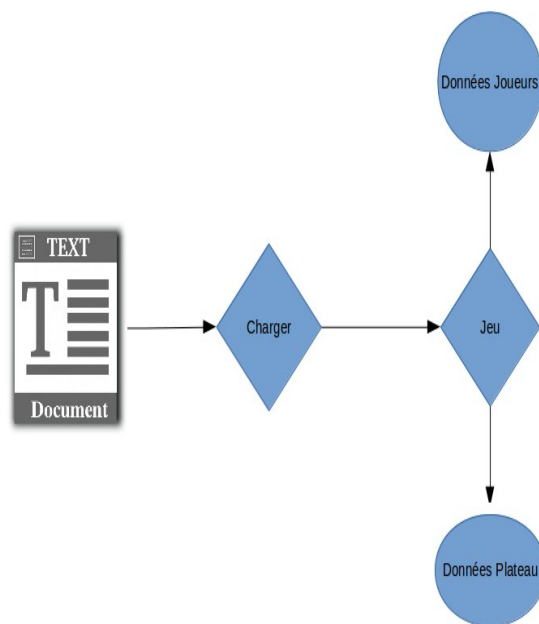
```
0 2 0 2 0 2 0 2 0 2
2 0 2 0 2 0 2 0 2 0
0 2 0 2 0 2 0 2 0 2
2 0 2 0 0 0 0 0 2 0
0 0 0 0 0 0 0 2 0 0
0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 1 0 1
1 0 1 0 1 0 0 0 1 0
0 1 0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0 1 0

ivan
1
17
0
1
0
mika
1
19
0
2
0
1
```

Sérialisation :



Désérialisation :



Intelligence artificielle :

L'intelligence artificielle utilisée dans ce programme se base sur un principe extrêmement simple : l'aléatoire. L'algorithme de jeu du joueur virtuel correspond presque exactement à celui du joueur réel, à l'exception que celui ci appelle des fonctions de choix de cases à la place de demander au joueur ce qu'il veut jouer. Ces fonctions de choix sont divisé en trois niveaux, facile, normal et difficile, selon le niveau de difficulté sélectionné en début de partie. Chacune de ces trois fonctions commence par sélectionner au hasard une des coordonnées qui lui sont transmises en paramètres dans un vecteur. A la suite de ce choix, la fonction facile et la fonction difficile détermine si ce choix entraînera l'élimination du pion courant au tour suivant. Si c'est le cas, le joueur facile prend cette décision, et joue de manière à être éliminer, tandis que le joueur difficile lui teste les autres possibilités, s'il en existe une moins risquée, il opte pour celle ci. Ces fonctions sont récursives mais pas infinies, puisque le choix précédent si il n'a pas été sélectionné est stocké dans un vecteur afin que l'ordinateur n'ait pas à réessayer avec le même pion encore et encore. La plupart du temps ces choix se limitant à deux ou trois coordonnées différentes, la complexité de ces algorithmes est négligeable. L'IA moyen lui opte pour la première option qu'il sélectionne aléatoirement. Ainsi, ces choix sont tantôt les mêmes que l'IA difficile, tantôt ceux de l'IA facile, ce qui en fait un adversaire de difficultés intermédiaire.

Cette tactique de jeu est cependant extrêmement limité et sûrement uniquement temporaire dans notre programme, car la stratégie de sacrifice de l'IA facile le mène souvent à la victoire, même face à l'IA difficile. Nous avons donc effectué des recherches quant à de nouvelles stratégies à intégrer à notre intelligence artificielle, afin de la rendre plus adaptée au jeu et aux niveaux de difficultés requis.

Interface graphique :

La partie graphique est composé de deux boucles essentielles au déroulement du jeu qui sont connectés par un compteur «cmpt» qui comptabilise les « clics » et permet de connaître l'instant où le choix d'une case à jouer a été fait. Ainsi cmpt=0 quand il n'y a aucune case de sélectionnée, lorsque l'on clique sur un pion pour le déplacer cmpt=1, enfin, quand on clique sur la case vide cmpt se réinitialise à zéro.

La boucle dans la classe Fenêtre utilise aussi un vecteur nommé «vect» qui contient les cases qui peuvent être joués.

On a choisit d'afficher les bord des cases en vert afin de savoir quelles cases ont la possibilité d'être jouées (dans une autre fonction appartenant a la classe PartieGUI). Les bords des cases sont initialement tous cachés.



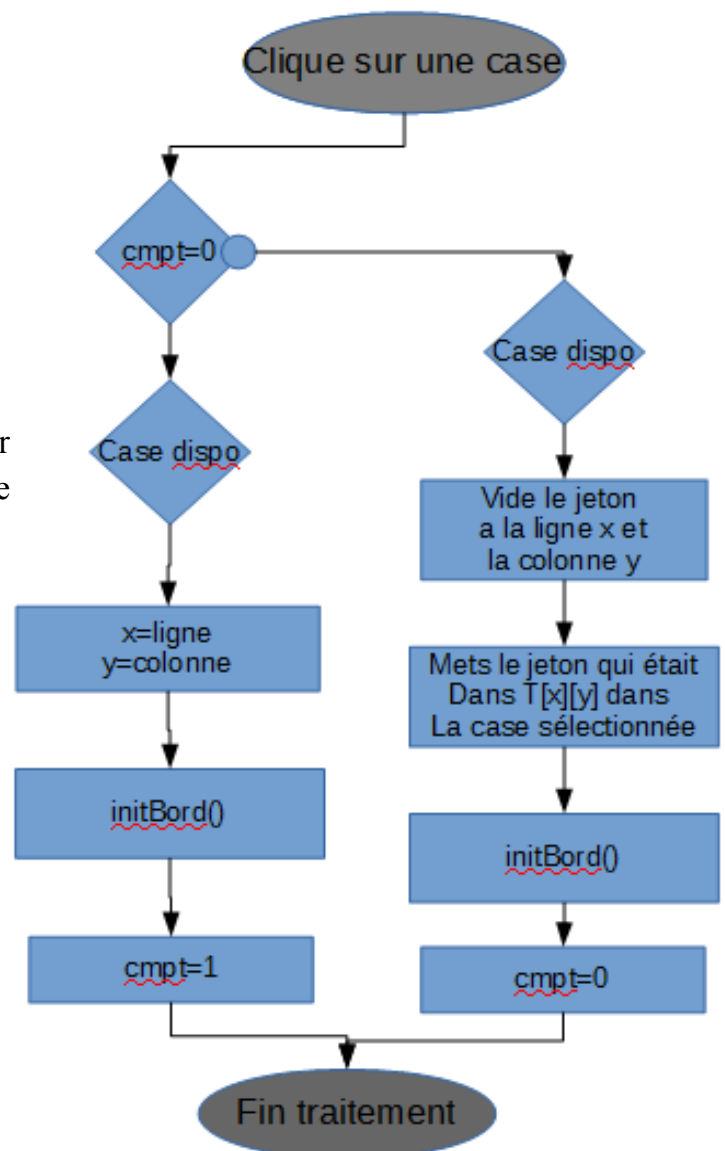
case disponible



case indisponible

Voici comment la classe Fenetre va s'occuper des déplacements des pions et qui se trouve dans la fonction «actionPerformed».

(Son algorithme en annexe 7.)



IV. Problèmes rencontrés et solutions apportées

- Programmation sous java:

La programmation sous java nous était inconnue, nous programmions notre jeu de dames en même temps que nous prenions nos premiers cours de java. Nous avons donc pris de l'avance sur l'apprentissage de java.

- Travail en groupe:

Cela a été notre premier obstacle, il a fallu s'adapter au travail des autres membres du groupe. Il nous a fallu faire des choix entre deux solutions pour un même problème proposées par deux membres du groupe. Nous avons réglé ce problème grâce au compromis de l'un d'entre nous et par la suite nous nous sommes distribués les tâches afin de parer à ce problème.

- Rafle

Durant les tests du jeu, nous avons rencontrés de nombreux problèmes avec les rafles, surtout celle concernant les dames. Nous avons été obligés de modifier de temps en temps certaines fonctions et même de recommencer certaines fonction depuis le début.

- Programmation événementielle:

Les problèmes concernant cette partie étant trop nombreux car nous touchions pour la première fois à la programmation événementielle nous ne citerons que les principaux. Le problème qui nous a le plus retardé fut l'actualisation des cases à jouer qui était trop lent pour pouvoir jouer normalement.

Nous avons recherché une solution sur internet et tenté de multiples résolutions pour se rendre qu'il fallait juste redessiner la case avec un repaint() quand la case devait changer de formes.

- Intelligence artificielle:

Nous n'avons pas eu assez de temps pour implémenter un IA vraiment intelligent. Les tactiques de jeu que nous avons implémenter pour l'instant sont extrêmement limité est sûrement uniquement temporaire dans notre programme, car la stratégie de sacrifice de l'IA facile le mène souvent à la victoire, même face à l'IA difficile.



VI. Conclusion

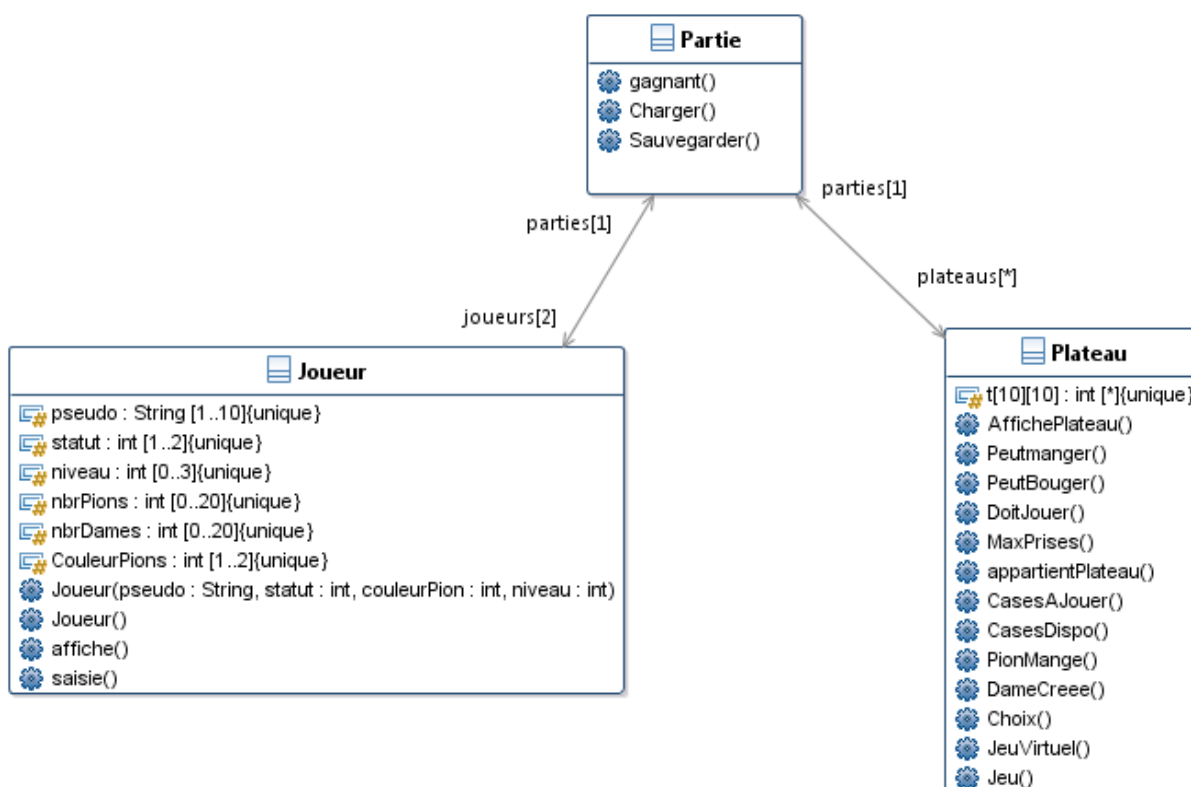
Par la suite, notre programme pourrait être adaptée pour une application sur mobile, ou sur le web, ce qui constituerait un défi nouveau pour notre groupe et élargirait nos capacités. Nous allons effectué des recherches quant à de nouvelles stratégies à intégrer à notre intelligence artificielle, afin de rendre le jeu contre l'ordinateur plus intéressant.

De notre coté, ce projet a développé nos compétences dans le domaine de la programmation et du travail en groupe. Nous avons développé un esprit de groupe et d'adaptation, qui nous sera utile lors de projet futurs dans notre vie professionnelle. Nous avons de plus grâce a ce projet approfondie nos compétence en programmation avec le langage Java. Nous avons eu la possibilités d'élargir l'arbre de nos compétences en appréhendant les interfaces graphique et les intelligences artificielles.

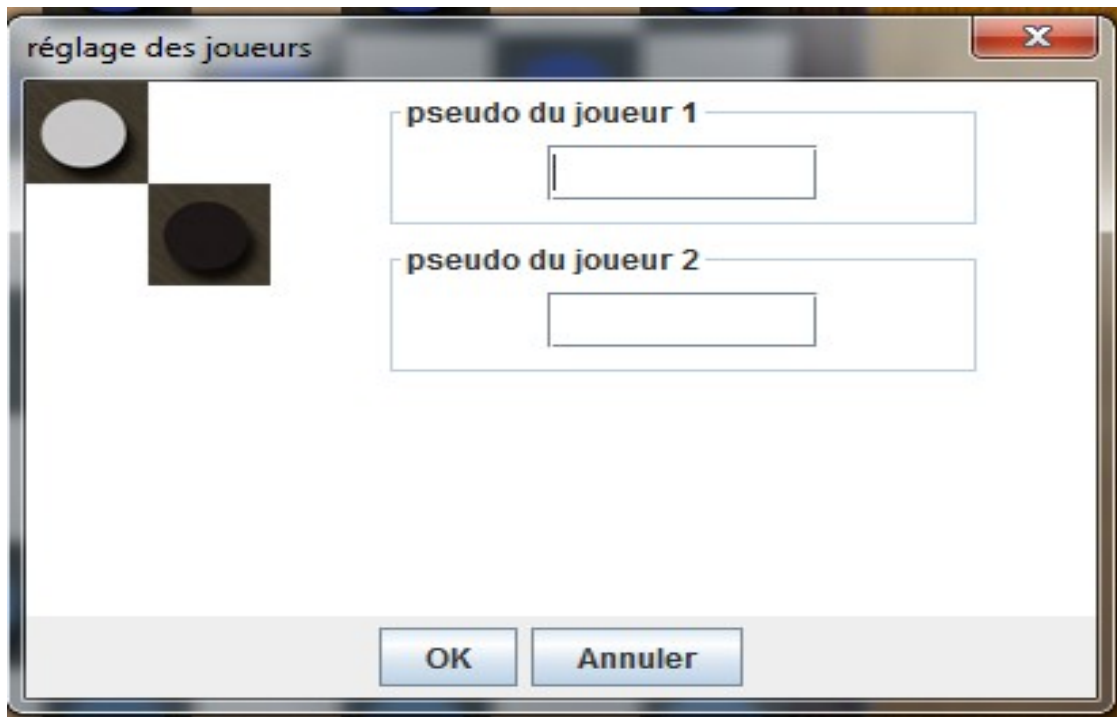
Nous avons enfin pu approfondir des notions vu en cours qui étaient mal assimilés comme les interface , l'héritage,etc et nous nous sommes aussi avancé sur des notions qui n'étaient pas encore au programme comme la programmation événementielle.

VII. Annexe

anx 1 :



anx 2 :

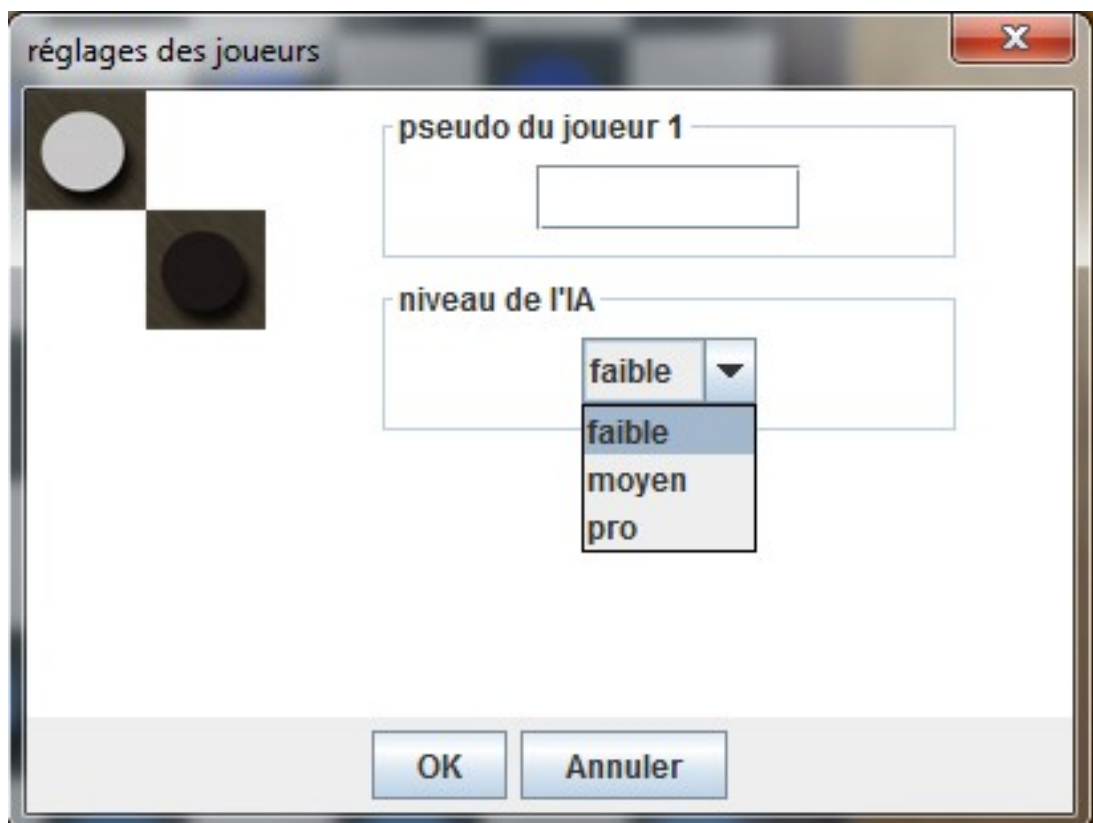


réglage des joueurs

pseudo du joueur 1

pseudo du joueur 2

OK Annuler



réglages des joueurs

pseudo du joueur 1

niveau de l'IA

faible

faible
moyen
pro

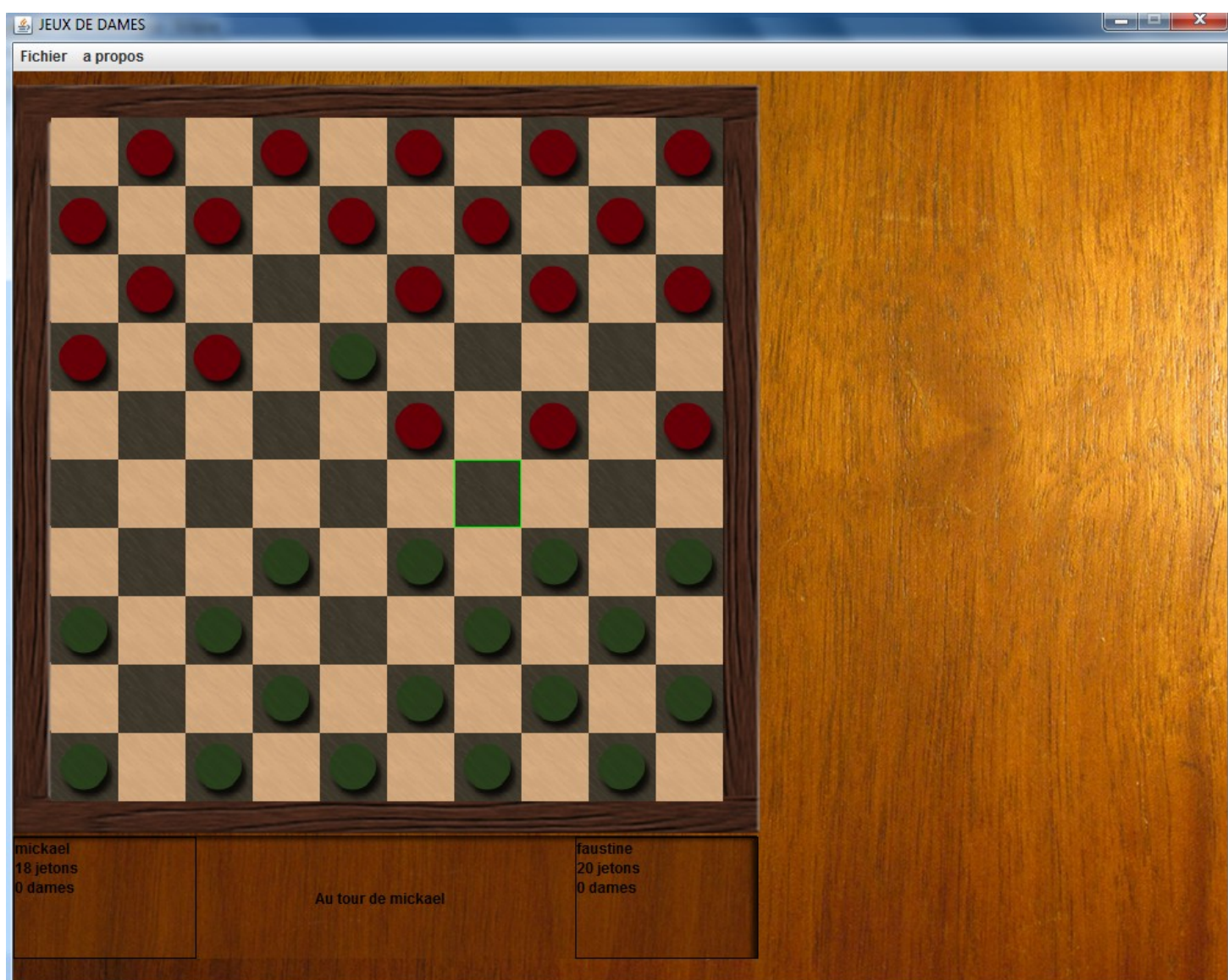
OK Annuler



anx 4 :

```
Java - jeuxDames/src/jeuxDames/PartieConsole.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
PartieConsole [Java Application] C:\Program Files\Java\jre1.8.0_40\bin\javaw.exe (20 avr. 2015 15:52:14)
0 0 0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0 1 0
mickael a: 20 pions et 0 dames
faustine a: 20 pions et 0 dames
mickael, voici les pions/cases que vous pouvez jouer:
5 2;
entrez votre choix de pion/case (la ligne puis la colonne):5
2
0 2 0 2 0 2 0 2 0 2
2 0 2 0 2 0 2 0 2 0
0 2 0 2 0 2 0 2 0 2
0 0 2 0 2 0 2 0 2 0
0 2 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0
0 0 0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0 1 0
[3, 0]
[3, 0]
mickael, voici les pions/cases que vous pouvez jouer:
3 0;
entrez votre choix de pion/case (la ligne puis la colonne):
```


anx 5 :



Annexe 6 : Algorithme de MaxPrises :

données: 3 entiers i, j : coordonnées de la case choisie et numéro du joueur courant cj , un vecteur d'entiers v qui contient les coordonnées des jetons mangés, un booléen qui indique si le pion choisi est une dame.

résultat: entier, nombre maximum de pièces ennemis mangés.

variables: a, b, c, d des entiers qui comptabilisent le nombre de pièces ennemis mangés selon les quatre directions possibles. m, n , deux autres entiers qui représentent les pions ennemis (selon si dame ou pion) et enfin un entier k , un compteur.

début:

$k=2$

initialisation de m et n en fonction du joueur courant.

tant que $k < 10$:

 si la case de coordonnées $i+k, j-k$ appartient au plateau :(étude des ennemis mangés dans la diagonale basse gauche.)

 si la case contient un 0 (si elle est vide):

 si la case $i+k-1, j-k+1$ contient un ennemi qui n'appartient pas au vecteur:

 on l'ajoute au vecteur v

$a = 1 + \text{MaxPrises}(i+k, j-k, \text{vecteur } v, \text{bool}, cj)$

 fin

 fin

 fin

 on réitère ce code pour les quatre directions possibles

$k++$

 si le boolean est faux k est affecté à 11.

fin

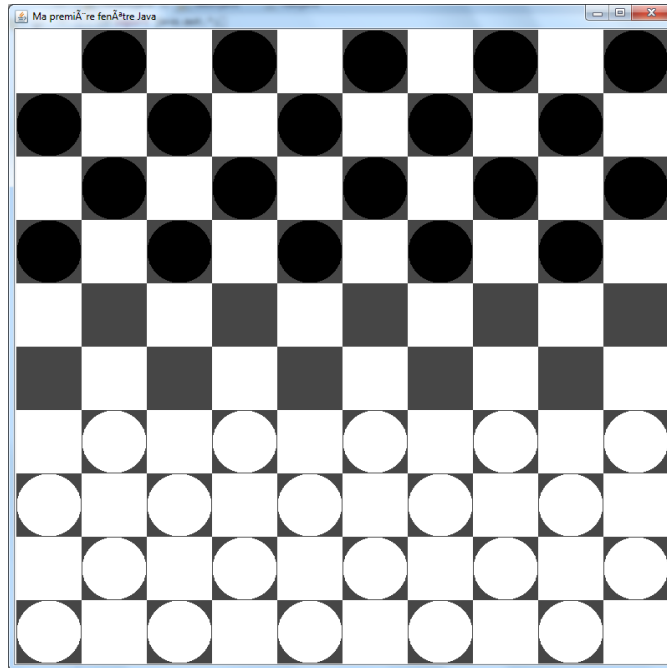
renvoyer $\max(a, \max(b, \max(c, d)))$;

fin

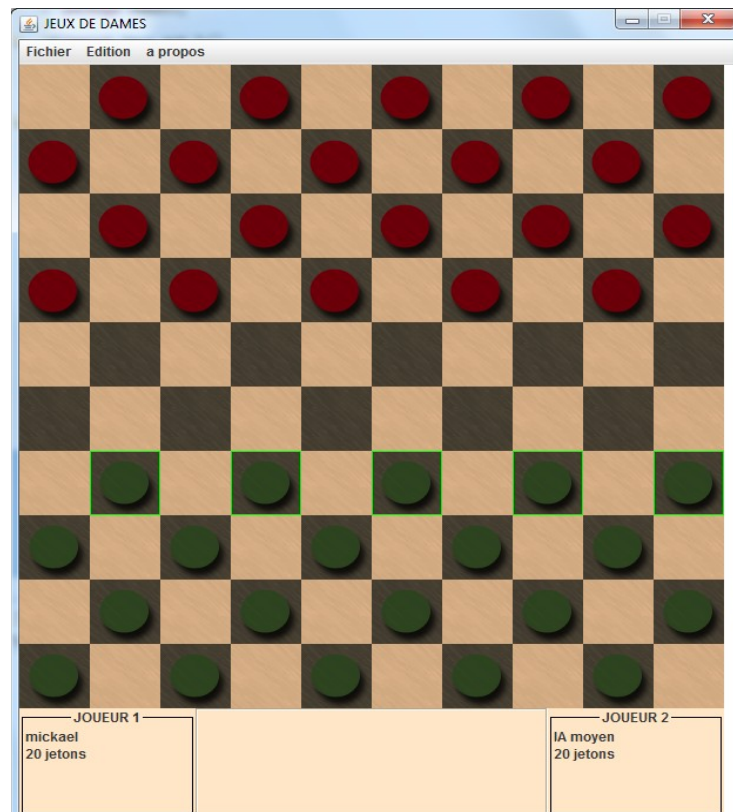
Annexe 7:

```
Si cmpt=0 alors
    si la case sélectionnée appartient à vect alors
        Ind[0]=jetonSélectionné.getLigne()
        Ind[1]=jetonSélectionné.getColonne()
        supprimer tous les éléments de vect
        initBord()//la fonction qui recache tous les bords des cases
    fin si
sinon
    si la case sélectionnée appartient à vect alors
        Ind[2]=jetonSélectionné.getLigne()
        Ind[3]=jetonSélectionné.getColonne()
        T[Ind[0]][Ind[1]].setC(0)//la case aux coordonnées Ind[0] et Ind[1] n'a plus de
        //jetons
        T[Ind[2]][Ind[3]].setC(la couleur de la case sélectionnée)
        initBord()
        supprime les éléments de vect
        cmpt=0
    fin si
fin si
```

évolution du design de l'interface graphique :
version 1 :



version 2 :



version finale :

