UNIVERSITE LIBRE DE BRUXELLES

INFO-F106

Rapport de projet

Pierre GÉRARD (000379259)

3/29/2013

Table des matières

Introduction

Afin d'appliquer et de développer les notions vues au cours de la première année de bachelier en sciences informatiques, il était demandé d'implémenter dans le cadre du cours INFO-F-106 une version légèrement simplifiée de la variante française du jeu de dames en Python3.Ce projet était divisé en quatre grandes parties.

Partie 1

La première partie consistait à réaliser le module *draughtsFunctions.py*qui correspond aux fonctions suivantes :

- ***initBoard(dimension)*** : qui initialise une matrice qui représente le plateau de jeu.
- ***printBoard(board, player)*** : qui affiche le plateau de jeu en fonction du joueur qui est en train de jouer.
- ***checkMove(board, i, j, direction, player)*** : qui vérifie si le déplacement passé en paramètre est autorisé.
- ***movePiece(board, i, j, direction)*** : qui déplace la pièce en position (i,j) suivant la direction passée en paramètre.
- ***checkEndOfGame(board, player)*** : qui vérifie si le joueur actuel peut encore effectuer un mouvement.

Lors de cette première partie, il était aussi demandé d'implémenter un module *draughts.py* qui permettait de faire interagir ces différentes fonctions et de proposer à l'utilisateur de jouer en ligne de commande sans effectuer de prises.

Partie 2

La deuxième partie consistait à modifier les fonctions de la première partie afin d'ajouter les règles (dames et prises) :

- ***printBoard(board, player)*** : affiche désormais aussi les dames.
- ***checkMove(board, i, j, direction, player)*** : renvoie désormais un code correspondant à l'erreur si le déplacement n'est pas correct ou qu'il n'y a pas d'erreur.
- ***movePiece(board, i, j, direction)*** : permet désormais de capturer un pion.

Il nous était aussi demandé de créer des fonctions supplémentaires :

- ***becomeKing(board, i, j)*** : qui transforme un pion en dame si ce dernier a atteint l'autre bout du plateau.
- ***capture(board, i, j)*** : qui retire le pion ou la dame capturé du damier.
- ***countFree(board, i, j, direction, player=None, length=0)*** : qui calcule le nombre de cases vides entre la case (i,j) et le premier obstacle dans la direction donnée.
- ***strerr(errCode)*** : qui transforme un code d'erreur en un message plus explicite.
- ***save(filename, board, player)*** : qui sauvegarde dans un fichier .dat le joueur actuel et le damier .
- ***load(filename)*** : qui permet de charger un fichier préalablement sauvegardé.

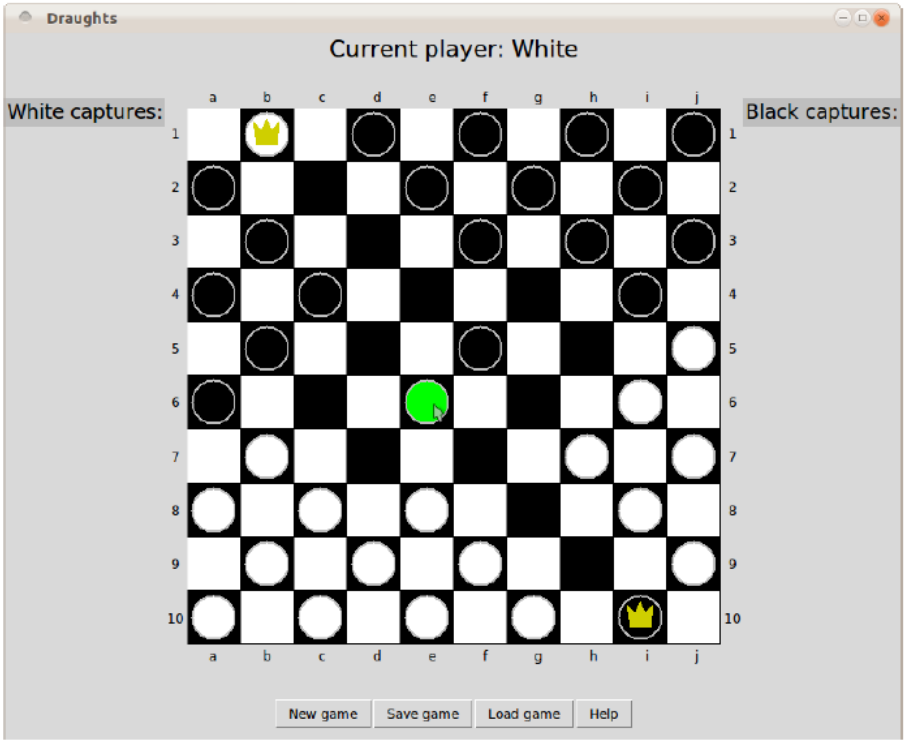
Partie 3

La troisième partie du projet consistait en la mise en œuvre à l'aide de la librairie *tkinter* d'une *interface graphique* qui permet à l'utilisateur de jouer au jeu de dames à l'aide de la souris.

Les éléments imposés étaient les suivants :

- Une zone affichant le damier avec tous les pions.
- Deux zones indiquant aux joueurs le nombre de pions adverses qu'ils ont pris.
- Une zone de statut indiquant quel joueur a le trait.
- Une zone avec quatre boutons permettant respectivement de commencer un nouveau jeu, sauvegarder la partie en cours, charger une partie sauvegardée et d'afficher une aide dans une fenêtre séparée.

L'interface ainsi créée devait se rapprocher le plus possible de la maquette suivante :



Et elle doit se comporter de la façon suivante :

- L'interface doit afficher quel joueur a le trait.
- Pour bouger un pion le joueur doit d'abord cliquer sur le pion qu'il veut bouger, ensuite il doit cliquer sur l'endroit où il veut aller (le pion bouge), finalement il doit recliquer sur le même pionpour signaler que son coup est fini.
- Si le joueur veut réaliser une prise multiple, il doit d'abord cliquer sur le pion qu'il souhaite utiliser,ensuite cliquer sur toutes les cases par lesquelles le pion doit passer pour faire toutes les prises (lepion bouge après chaque "clic") et ensuite cliquer sur le même pion pour signaler que son tour est fini.
- Dans le cas d'une erreur d'input, l'interface doit signaler l'erreur à l'utilisateur en précisant la nature de l'erreur.

Partie 4

La quatrième et dernière partie de ce projet consistait à mettre en place une *intelligence artificielle*permettant à un joueur humain de jouer contre un ordinateur ayant un très faible niveau.

L'algorithme permettant de réaliser les prises considère un nombre déterminé de pions et utilise la stratégie suivante :

if Une prise d'un pion adverse est réalisable ? **then**
    Prendre le pion adverse
    while Une prise d'un pion adverse est réalisable avec le pion choisi précédemment **do**
        Prendre le pion adverse
    end while
else if Un pion de l'ordinateur peut se déplacer sans être mangé au coup suivant ? **then**
    Bouger le pion choisi
else
    Bouger un autre pion

Il doit être implémenté uniquement pour fonctionner avec l'interface graphique.

Difficultés et solutions apportées

Fichier de configuration

Afin de faciliter la configuration, il est demandé d'écrire un fichier de configuration. Ce fichier contient une vingtaine de variables locales qui définissent les caractères représentant les pions, les erreurs, le nombre de pions considérés dans l'intelligence artificielle et les différents modes de jeu.

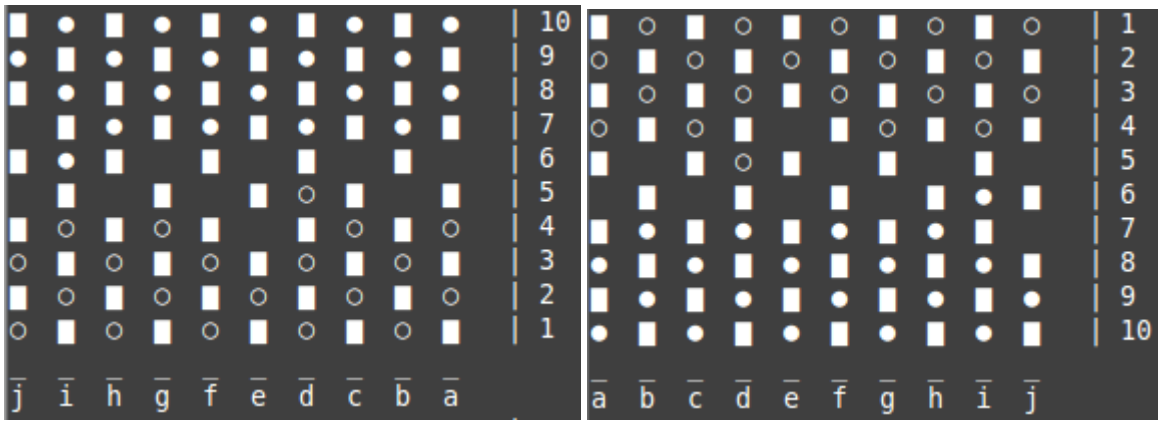
Fonctions de base

Initialisation du damier

Le damier est composé de 10 cases sur 10 et chaque joueur possède 20 pions situés sur les cases blanches jouables. Le damier est représenté par une liste contenant 10 sous-listes qui correspondent aux lignes. Ces sous-listes contiennent chacune 10 entiers qui correspondent aux pions. Une valeur de 0 équivaut à une case vide, une valeur négative à une pièce noire et une valeur positive à une pièce blanche. Le damier est donc une matrice.

Impression en ligne de commande du damier

Pour faciliter la lecture du damier, il est demandé d'écrire une fonction qui va imprimer un tableau de la façon suivante pour respectivement le joueur noir et le joueur blanc :



L’impression pour le joueur blanc est aisée car la matrice contenant le damier se présente de la même manière que celle imprimée. Par contre pour le joueur noir il faut appliquer une symétrie et renverser toutes les lignes et les colonnes.

Vérification d’un coup

L'implémentation de la vérification de la validité d'un coup et du respect des règles est la partie de la structure de base la plus difficile à réaliser. Il faut gérer douze erreurs différentes. La fonction checkMove doit vérifier que :

- Le format de direction entré par le joueur est correct et est bien une des 4 possibilités.
- Il y a une pièce sur le damier à l'endroit qui correspond aux coordonnées entrées par le joueur.
- La pièce que le joueur essaie de déplacer lui appartient.
- Les nouvelles coordonnées après le déplacement souhaité par le joueur se situent bien sur le damier.
- Sur la case correspondant aux coordonnées que donnerait le déplacement, il n'y a pas de pièce appartenant au joueur, ce qui rendrait le déplacement impossible.
- Si sur la case correspondant aux coordonnées entrées par le joueur, il y a un pion, alors la longueur du déplacement doit être d’une case. On peut aussi noter que cette erreur peut ne jamais se produire si on ne demande pas la longueur du déplacement au joueur quand il essaie de déplacer un pion.
- Si le déplacement entraine une capture, il faut que la case située après la pièce que le joueur essaie de capturer ne se trouve pas en dehors du damier.
- Le joueur n'essaie pas de sauter plusieurs pièces en un déplacement. Pour cela, il faut vérifier qu’après la première capture possible il y a une case libre.
- Si le joueur essaie de déplacer un pion en arrière, il doit y avoir capture.
- Si le joueur poursuit une rafle, le mouvement qu'il veut effectuer doit capturer une pièce. La même erreur est renvoyée si le joueur veut continuer à jouer alors qu'il n'a pas capturé lors de son premier déplacement. Cependant ce dernier cas ne se produit jamais si la fonction principale empêche le joueur de rejouer s’il ne capture pas de pièce.
- Si le joueur déplace une dame, elle ne passe pas au-dessus de plusieurs pions. Pour cela, la fonction fait appel à une autre fonction *countFree* qui renvoie le nombre de pièces jusqu'au premier obstacle (bord ou pion). Ensuite, elle vérifie que ce nombre n'est pas inférieur à la longueur du déplacement.

Sinon la fonction renvoie qu'il n'y a pas d'erreurs et le joueur peut effectuer son mouvement. Elle renvoie en réalité un code d'erreur numérique différent pour chaque erreur. Ce code numérique sera traduit au joueur par la fonction *sterr*.

Déplacement d'un pion et capture

Lorsque qu'un déplacement est vérifié, le programme peut modifier la matrice représentant le damier pour que le déplacement soit effectué. C'est la fonction movePiece qui va s'en occuper. Elle va ensuite renvoyer les coordonnées de la destination du pion et les coordonnées de la pièce capturée si il y en a une et None sinon. Ensuite s'il y a eu capture, on va enlever la pièce du damier.

Détection de la fin de la partie et du gagnant

Une partie est terminée lorsqu'un joueur ne sait plus bouger ou n'a plus de pièces. C'est alors son adversaire qui remporte la partie. Pour cela, il faut détecter si le joueur passé en paramètre possède encore une pièce ou s'il sait faire un mouvement. On effectue cette dernière tâche en testant tous les mouvements jusqu'à ce qu'elle trouve une possibilité qui fonctionne s'il y en a une. Ensuite, on renvoie ce résultat. La fonction initiale de vérification va ensuite faire de même pour l'autre joueur et analyser les différents résultats. On pourra ensuite renvoyer le résultat; le joueur gagnant, un match nul ou bien une partie pas encore terminée.

Sauvegarde et chargement

Pour rajouter des fonctionnalités au jeu, il est aussi demandé d'implémenter deux fonctions qui vont sauvegarder et restaurer la partie en cours. Ces deux fonctions vont écrire et lire sur un fichier dont le format *.dat* est imposé, le joueur qui doit jouer le prochain coup, la taille du damier, le mode de jeu et le damier lui-même.

Jeu en ligne et commande et classe principale

Le jeu en ligne de commande respecte le pseudocode suivant :

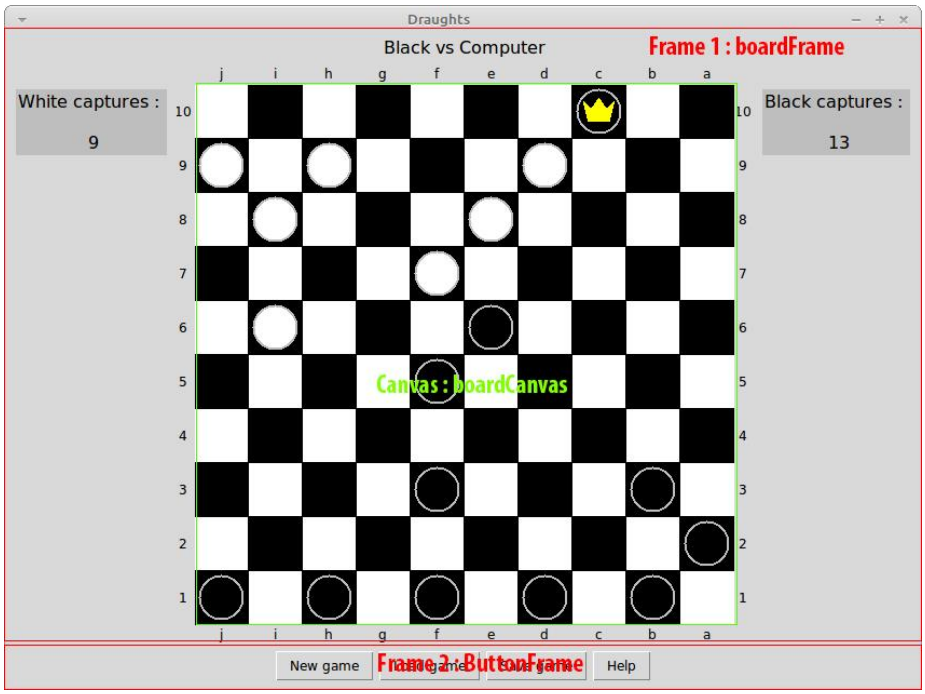
```
Initialiser le damier
Joueur = joueur blanc
Tant que le jeu n'est pas fini:
   Imprimer le damier et le joueur qui a le trait
   choix = ce que le joueur veut faire
   Si choix == déplacer une pièce :
      Tant que le déplacement n'est pas valide :
         Demander le premier déplacement souhaité.
      Effectuer le déplacement
      Si pas de capture:
         Indiquer au joueur pas de capture
      Sinon:
         Demander au joueur si il veut poursuivre sa rafle
         Tant que le joueur souhaite poursuivre sa rafle:
            Demander le déplacement et vérifier sa validité
            Effectuer le déplacement si il est valide
            Redemander joueur si il veut poursuivre sa rafle
      Passer à l'autre joueur
      Vérifier la fin du jeu
   Sinon si choix == proposer un match nul à l'adversaire:
      Demander à l'autre joueur si il accepte
   Sinon si choix == sauvegarder :
      Sauvegarder une partie
   Sinon si choix == restaurer :
      Restaurer une partie
```

Cet algorithme est implémenté dans la classe située dans le fichier *draughts.py*.Cette dernière est découpée de façon à ce que l'interface graphique puisse l'utiliser.

Interface graphique

Création de la maquette

Avant de commencer, il faut s'imaginer une maquette qui représentera l'interface graphique. La structure globaleest la suivante :



Elle contient :

- Deux frames différentes contenues dans la frame principale (masterFrame) ce qui permet d'avoir des alignements indépendants entre chaque Frame.
- Un canevas principal contenant les formes géométriques qui correspondent aux pions, aux dames et aux carrés noirs.
- Des éléments secondaires : 4 petits canevas qui permettent d'indiquer les coordonnées, 2 labels qui indiquent les captures et un label qui indique le joueur actuel.

Alignements des éléments

Après avoir défini tous les éléments que l'on utilisera, il faut les aligner. La méthode *grid* semble être un choix idéal car elle permet de résoudre le problème de positionnement en plaçant les éléments sur un quadrillage. La raison pour laquelle on a créé deux frames se situe ici et est d'obtenir des alignements indépendants (chaque frame a un quadrillage différent).La frame boardFrame est de dimension 4x5 et la frame ButtonFrame sera de dimension 1x4.

Mise à jour

Une mise à jour de l'interface graphique consiste a effectuer les opérations suivantes :

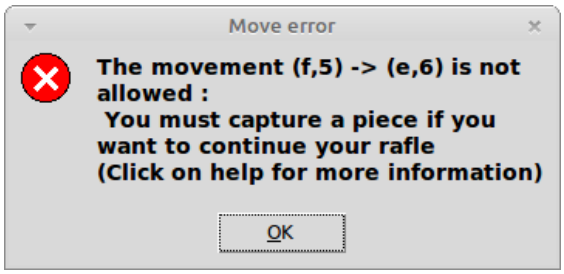
- Retirer tous les pions du canevas et les replacer en fonction du joueur et si besoin, inverser les coordonnées affichées sur les bords du canevas.
- Modifier le champ du joueur actuel.
- Mettre à jour les captures.

Gestion des clics

Il est facile grâce à event/bind de capturer les coordonnées des clics. Cependant un des difficultés de l'interface graphique consistait à convertir les coordonnées des clics en données utilisables par les fonctions précédemment définis. Il a donc fallu les comparées pour obtenir une direction et vérifier que le point de départ et d'arrivée d'un pion se situe bien sur une même diagonale.

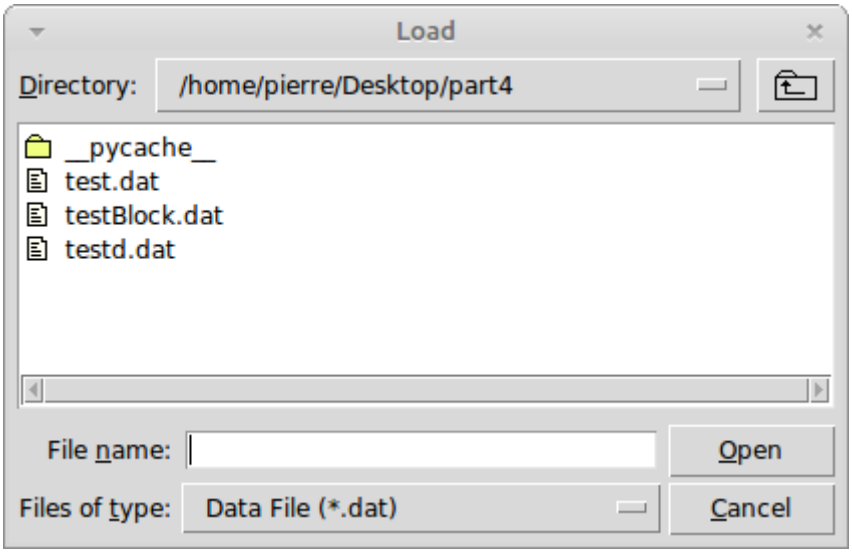
Gestion des erreurs

Les erreurs lors d'un déplacement s'affichent sous forme de pop-up de la manière suivante grâce à la méthode *messagebox.showerror*:



Sauvegarde et chargement

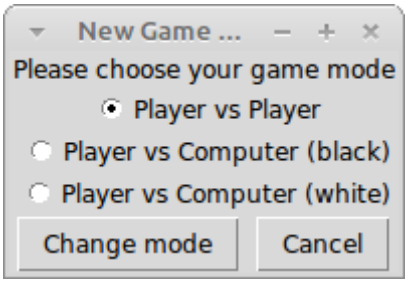
La sauvegarde et le chargement d'une partie peuvent se faire facilement avec la méthode *filedialog.askopenfilename* de tkinter qui permet d'obtenir la fenêtre suivante :



Cette méthode permet de récupérer le nom et le chemin du fichier qu'il suffit ensuite d'utiliser dans les fonctions load et save précédemment définies.

Choix d'une nouvelle partie

Pour permettre à l'utilisateur de commencer une nouvelle partie il faut qu'il ait le choix entre plusieurs options possibles. Pour cela on crée un pop-up avec des RadioButton de la manière suivante :



Intelligence artificielle

Modification à réaliser

L'implémentation d'une intelligence artificielle nécessite les modifications suivantes :

- Laisser le choix au joueur du mode de jeu dans lequel il veut jouer lorsqu'il commence une nouvelle partie, contre un autre joueur, contre l'ordinateur qui possède les pions noirs ou contre l'ordinateur qui possède les pions blancs.
- Modifier les fonctions de sauvegarde et chargement de manière à conserver le mode de jeu.
- Modifier l'affichage du joueur actuel pour y indiquer que l'on joue contre l'ordinateur.

Compréhension de l'algorithme

L'algorithme donné n'est pas facile à comprendre et peut-être interprété de diverses manières. La lecture qui en a été faite est la suivante :

1. On commence par choisir aléatoirement 10 pions et on regarde si un de ces pions peut capturer. Si c'est le cas on effectue le maximum de captures possibles avec ce pion.
2. Sinon on choisit aléatoirement 10 pions et on regarde si un de ces pions peut effectuer un déplacement sans se faire capturer au tour d'après. Si c'est le cas on le déplace. Notons bien que l'on parle ici de déplacement et que les prises ne sont donc pas exclues.
3. Sinon on effectue un autre déplacement aléatoirement.

Liste des pions de l'IA

Pour limiter le nombre d'opération de l'IA, il est demandé de conserver dans une liste la position de tous les pions de l'ordinateur. Cependant ces pions peuvent disparaitre lorsque l'utilisateur se déplace. Il faut donc créer une méthode qui vérifie que des pions n'ont pas disparu avant de vérifier s'ils peuvent se déplacer.

Mouvements avec capture

Il est aisé de détecter si un mouvement va conduire à une capture. Il suffit de regarder s'il est autorisé et s'il y a un pion dans le chemin. Il en va de même pour poursuivre une rafle.

Mouvement sans capture immédiate

Ce bout d'algorithme est de loin la partie la plus complexe de l'IA, il faut considérer que le mouvement peut induire une capture. Il faut donc trouver le point de destination du pion et ensuite regarder s'il y a un pion adverse dans une des directions. Si c'est le cas il faut ensuite regarder s'il y a une case libre dans la direction opposée. Si c'est aussi le cas alors l'adversaire pourra capturer le pion et le mouvement n'est pas considéré comme un déplacement ou le pion pourra être mangé au coup suivant.

Mouvement aléatoire

Pour trouver un mouvement aléatoire, il suffit de regarder s'il est autorisé.

Implémentation dans l'interface graphique

Après avoir créé une classe qui permet de simuler un joueur, il faut l'inclure dans le code. Le meilleur endroit pour le faire est lorsque l'on change de joueur dans la classe principale du programme.

Conclusion

Le programme implémente bien une version complète du jeu de dames, version fonctionnelle et facile à utiliser.

Les quatre parties nous ont permis de développer nos connaissances et en particulier la partie 3 où nous avons dû apprendre à créer une interface graphique par nous-mêmes.

Il est aussi très satisfaisant de voir ce dont on est capable et de constater que l'on a acquis suffisamment de connaissances pour développer un programme complet.