FACULTÉ DES SCIENCES
DÉPARTEMENT D'INFORMATIQUE

INFO-F106 : PROJETS D'INFORMATIQUE 1

Vandy Berten Gwenaël Joret Jean-Sébastien Lerat Liran Lerman
Catharina Olsen Nikita Veshchikov

version 1.1 - 5 novembre 2012

Note : Les modifications par rapport à la première version de l'énoncé apparaissent en bleu.

1 Introduction

1.1 Idée générale

Le but de ce projet est de réaliser, en Python 3, une version légèrement simplifiée du jeu de dames, plus précisément de sa variante française. Il se joue sur un damier de 10 cases par 10, en n'utilisant que les cases noires, démarre avec 40 pions (20 noirs et 20 blancs). Les règles simplifiées que nous utiliserons sont définies plus bas. La principale simplification consiste à ne pas rendre les prises obligatoires. Les deux premières étapes consisteront en la mise en place de la structure de données et en la vérification du respect des règles. La troisième étape ajoutera au jeu une interface graphique. Dans la quatrième et dernière étape, on donnera à un joueur la possibilité d'affronter l'ordinateur, en implémentant une stratégie très simple.

1.2 Objectifs pédagogiques

Ce projet « transdisciplinaire » permettra de solliciter les compétences des étudiants selon différents aspects.

- Des connaissances vues aux cours de programmation, langages, algorithmique ou mathématiques seront mises à contribution, avec une vue à plus long terme que ce que l'on retrouve dans les divers petits projets de ces cours. L'ampleur du projet requerra une analyse plus stricte et poussée que celle nécessaire à l'écriture d'un projet d'une page ainsi qu'une utilisation rigoureuse des différents concepts vus aux cours ;
- Des connaissances non vues aux cours seront nécessaires et les étudiants seront invités à les étudier par eux-mêmes, soit de façon totalement autonome, soit grâce aux « tuyaux » fournis par l'équipe encadrant le cours. Il s'agit entre autres d'une connaissance de base des interfaces graphiques en Python ;
- Des compétences de communication seront également nécessaires : un peu avant Noël et un peu avant Pâques, les étudiants remettront un rapport expliquant leur analyse, les difficultés rencontrées et les solutions proposées. Une utilisation correcte des outils de traitement de texte (utilisation des styles, homogénéité de la présentation, mise en page, etc.) sera attendue de la part des étudiants. Une orthographe correcte sera bien entendu exigée ;
- En plus des deux rapports, une présentation orale, avec « transparents » ou « slides » (produits par exemple avec PowerPoint ou OpenOffice Impress) sera organisée, ainsi qu'une démonstration de l'outil développé. À nouveau, on attendra des étudiants une capacité à présenter et à vulgariser leur projet (c'est-à-dire rendre compréhensible pour des non informaticiens, ou en tout cas pour des étudiants ayant eu le cours de programmation mais n'ayant pas connaissance de ce projet-ci).

En résumé, on demandera aux étudiants de montrer qu'ils sont capables d'appliquer des concepts vus aux cours, de découvrir par eux-mêmes des nouvelles matières et enfin de communiquer de façon scientifique le résultat de leur travail.

1.3 Règles du jeu

Les règles du jeu présentées ci-dessous correspondent à ce à quoi vous arriverez après les deux premières étapes. Les aspects pratiques ne sont pas abordés ici ; ils le seront dans les sections suivantes. Les règles complètes du jeu de dames sont disponibles sur le site de la Fédération Française du Jeu de Dame (`http://www.ffjd.fr/`). Rappelons que nous considérons ici une version simplifiée, et que certaines des règles soit ne sont pas d'usage dans ce projet, soit ont été modifiées.

1.3.1 Initialisation

Le damier est composé de 10 cases sur 10, et seules les cases noires peuvent être utilisées. Chaque joueur a sur la gauche de la première ligne (par rapport à sa vue du damier) une case noire. Il place 20 pions de sa couleur (noir ou blanc) sur les cases noires des quatre premières rangées qu'il a face à lui. Il reste donc deux lignes libres au milieu du damier.

Le joueur ayant les pions blancs est le premier à jouer. On dit qu'il *a le trait*.

1.3.2 Déplacement

Les joueurs jouent en alternance, et déplacent un pion par tour. Un pion normal (nous reviendrons sur les « dames » ultérieurement) peut se déplacer d'une case, en diagonale (de façon à rester sur une case noire), uniquement vers l'avant (c'est-à-dire vers l'autre joueur), pour autant que la case de destination soit libre et située à l'intérieur du damier.

1.3.3 Prises

Lorsqu'un pion d'une couleur A trouve un pion de couleur B en diagonale sur un de ses quatre coins (vers l'avant OU vers l'arrière), et que la case suivante dans la même direction est libre, le joueur A peut déplacer son pion sur la case libre (il fait donc un saut de deux cases), et « prendre » le pion B, qui est donc retiré du damier. Si, à sa nouvelle position, il peut à nouveau effectuer une prise, il peut l'effectuer immédiatement, autant de fois que possible. On parle alors de *rafle par un pion*.

Nous modifierons ici les règles internationales en deux points, pour simplifier l'implémentation :
- Il n'est pas obligatoire de prendre un pion lorsque c'est possible, ni de faire la prise maximale ;
- Lors d'une rafle, le pion pris est retiré du damier directement, et non pas à la fin de la prise multiple.

1.3.4 Dames

Lorsqu'un pion arrive (à la fin d'un coup, qu'il y ait eu prise ou non) sur la dernière ligne du damier (c'est-à-dire celle qui se trouve le plus près du joueur adverse), il devient une « dame » (*king* dans la version anglaise). Sur un jeu de plateau classique, on la représente en superposant deux pions.

Lors de ses déplacements normaux, la dame peut non seulement également se déplacer vers l'arrière, mais peut aussi sauter un nombre arbitraire de cases libres. Lors de la prise, elle peut également franchir plusieurs cases vides avant le pion visé, mais doit « atterrir » sur la case libre juste après sa cible (contrairement aux règles internationales, autorisant d'atterrir n'importe où après le pion pris). La dame ne peut donc passer au-dessus d'aucune autre pièce que sa cible.

1.3.5 Fin de partie

La partie est terminée lorsqu'une des trois situations suivantes se produit :
- Un des deux joueurs n'a plus aucun pion. Son adversaire remporte alors la partie ;
- Plus aucun mouvement n'est possible pour aucun des joueurs. Il s'agit d'un match nul ;
- Plus aucun mouvement n'est possible pour le joueur qui a le trait, alors que son adversaire peut encore effectuer des mouvements. Le joueur immobilisé perd alors la partie.

1.4 Consignes

Il est vivement conseillé de relire ces consignes avant chaque remise d'une partie de projet !

- Les « consignes de réalisation des projets » (cf. `http://www.ulb.ac.be/di/consignes_projets_INFO1.pdf`) seront d'application pour ce projet individuel. On lira ces consignes en considérant chaque partie de ce projet d'année comme un projet à part entière. Relisez-les régulièrement également ;
- Le projet sera organisé autour de quatre grandes étapes. Il y aura également un rapport intermédiaire à remettre au milieu du projet ainsi qu'un rapport final à la fin. Chaque partie comptera pour 20 points et une cinquième partie (incluant le rapport, la démo et la présentation) comptera également pour 20 points, ce qui nous fait un total de 100 points ;
- Chaque partie devra être remise sur deux formats :
  - via l'Université Virtuelle (UV), sur `http://uv.ulb.ac.be`, qui contient une tâche pour chaque partie,
  - sur papier, au secrétariat étudiants du Département d'Informatique, une version imprimée du code source de chacun des fichiers envoyés par courriel ;
- Après chaque partie, un correctif sera proposé. Il vous est loisible de continuer sur base de ce correctif, mais nous conseillons aux étudiants de continuer avec leur travail en tenant compte des remarques qui auront été adressées ;
- En plus des quatre parties à remettre, les étudiants devront préparer une « démo » en salle machine de cinq minutes par personne, aux alentours de la semaine de cours 19. Par ailleurs, une défense orale de dix minutes par personne sera organisée aux alentours de la semaine de cours 20 ;
- L'ensemble du projet sera à réaliser en Python 3 ;
- En cas d'incompréhension sur un point de l'énoncé, veuillez contacter **le titulaire du cours ou la personne de contact de la partie concernée**, et non votre assistant de TP ;
- **Il n'y aura pas de seconde session pour ce projet !**

1.5 Organisation

Titulaires :

- Premier semestre : Vandy Berten – vandy.berten@ulb.ac.be – N8.111
- Second semestre : Gwenaël Joret – gwenael.joret@ulb.ac.be – 08.111

Assistants :

- Jean-Sébastien Lerat – jean-sebastien.lerat@ulb.ac.be – O8.212
- Liran Lerman – liran.lerman@ulb.ac.be – N8.212
- Catharina Olsen – catharina.olsen@ulb.ac.be – O8.213
- Nikita Veshchikov – nikita.veshchikov@ulb.ac.be – N8.213

2 Partie 1 : Création des structures de base

Cette partie contient quatre tâches principales : la création du damier, le déplacement tour par tour (sans prises), la vérification d'un coup et la détection de la fin de partie par blocage.

2.1 L'initialisation

Le damier sera représenté par une matrice 10×10. La matrice contient un zéro s'il n'y a pas un pion présent sur la case correspondante (qu'elle soit noire ou blanche), les pions blancs seront représentés par 1 et les pions noirs par -1. Vous allez écrire une fonction

```
initBoard(dimension)
```

dont le paramètre `dimension` correspond à la taille du damier. Cette fonction renverra la matrice initiale :

```
myboard=initBoard(10)
```

Après cette initialisation, vous aurez créé une matrice telle que celle représentée en Figure 1.
Si le paramètre `dimension` n'est pas égal à 10, on remplira la matrice de façon à garder deux lignes vides au centre, si `dimension` est un nombre pair, trois lignes vides dans le cas contraire. La méthode renverra `False` si elle est appelée avec un paramètre inférieur à 4.

										i :
	0	-1	0	-1	0	-1	0	-1	0	-1	0
	-1	0	-1	0	-1	0	-1	0	-1	0	1
	0	-1	0	-1	0	-1	0	-1	0	-1	2
	-1	0	-1	0	-1	0	-1	0	-1	0	3
	0	0	0	0	0	0	0	0	0	0	4
	0	0	0	0	0	0	0	0	0	0	5
	0	1	0	1	0	1	0	1	0	1	6
	1	0	1	0	1	0	1	0	1	0	7
	0	1	0	1	0	1	0	1	0	1	8
	1	0	1	0	1	0	1	0	1	0	9
j :	0	1	2	3	4	5	6	7	8	9	

FIGURE 1 – Matrice retournée par `initBoard(10)`

Ensuite il faut écrire une fonction

```
printBoard(board,player)
```

qui permet d'afficher le damier tel que les cases blanches sans pion correspondent à un carré noir [1] « ■ » (le caractère UTF-8 « $0x2587$ », que l'on peut afficher en Python à l'aide de l'instruction `print("\u2587")`) et les cases noires sans pion à une simple espace [2] (le caractère UTF-8 « $0x0020$ »). De plus, les pions blancs seront décrits par « • » (le caractère UTF-8 « $0x25CF$ ») et les pions noirs par « ∘ » (le caractère UTF-8 « $0x25CB$ »). Par ailleurs, on vous demande d'afficher les numéros des lignes et colonnes pour faciliter l'interaction avec l'utilisateur. Enfin, un appel à cette fonction affichera la matrice initiale comme présentée dans la Figure 2. Le paramètre `player` spécifie quel joueur a le trait (soit 1, soit -1). Le damier doit être affiché de sorte que les pions du joueur ayant le trait se déplacent vers le haut. La Figure 2 est créée avec l'instruction `printBoard(initBoard(10),1)`, l'instruction `printBoard(initBoard(10),-1)` affichera la Figure 3.
Nous attirons l'attention des étudiants sur les deux points suivants :

1. Nous supposons que vous exécuterez votre code Python dans un terminal ou un environnement « négatif », à savoir en écriture blanche sur fond noir. Dès lors, le symbole dénommé « carré noir » « ■ » sera donc un carré blanc sur fond noir.
2. Petite note culturelle : l'espace, en tant que caractère typographique, est un mot féminin. On a donc une espace entre deux mots. Par contre, l'espace — dans le sens « distance » — qui sépare deux lignes de texte, ou l'espace — dans le sens « univers » — qui nous entoure, est un mot masculin. On peut donc avoir un grand espace entre deux lignes de texte.

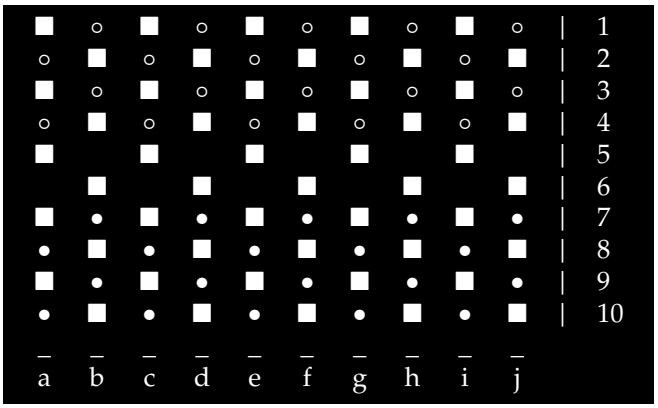

FIGURE 2 – Matrice affichée par `printBoard(initBoard(10),1)`

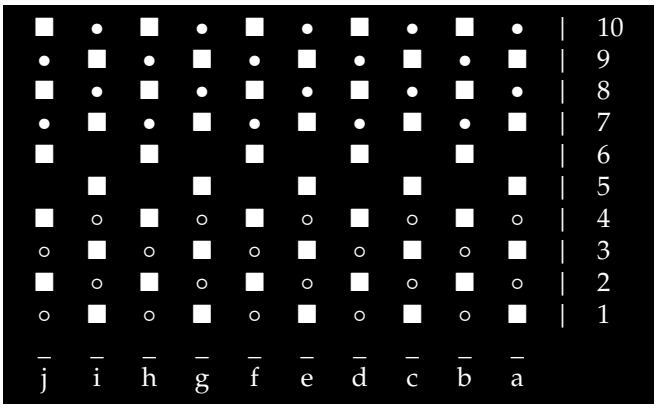

FIGURE 3 – Matrice affichée par `printBoard(initBoard(10),-1)`

- En fonction du terminal et de l'environnement que vous utilisez, il se peut que les caractères que nous avons choisis pour représenter des carrés soient plutôt rectangulaires. Rien ne vous empêche, durant le développement, de modifier le fichier `config.py` pour choisir un symbole qui soit mieux adapté à votre environnement, par exemple le « 0x25A0 » pour le carré. Sachant que vous ne nous soumettrez pas ce fichier, nous conserverons, à la correction, l'aspect original de l'affichage.
- Le système « DOS » de Windows supporte assez mal l'encodage UTF-8. Pour les utilisateurs de ce système d'exploitation, il est dès lors conseillé d'utiliser IDLE ou un autre système permettant d'afficher des caractères UTF-8.

2.2 Le déplacement d'un pion

Les règles pour le déplacement d'un pion sont faciles : un pion peut se déplacer seulement en avant. Par exemple, le pion blanc qui se retrouve à la position $(d, 7)$ peut soit aller à gauche vers la position $(c, 6)$ soit à droite vers $(e, 6)$. Les directions seront passées en utilisant le paramètre `direction` où « `L` » signifiera gauche et « `R` » signifiera droite.

```
movePiece(board, i, j, direction)
```

Les paramètres `i` et `j` correspondent à la position d'un pion dans la matrice originale, comme en Figure 1. Par exemple, pour effectuer le premier mouvement du pion en position $(d, 7)$ vers $(c, 6)$, il faut appeler la fonction avec les paramètres suivants :

```
movePiece(board,6,3,'L')
```

Notez que pour connaitre la direction dans laquelle le mouvement se fait réellement, il faut savoir quel pion on déplace. Mais il n'est pas nécessaire de le passer en paramètre, car on peut le trouver en position `board[i][j]`. Notez également que les joueurs utiliseront des coordonnées de type « a, 5 » (où la première coordonnée représente la colonne et va de « a » à « j », alors que la seconde coordonnée représente la ligne, et va de « 1 » à « 10 »), alors que les fonctions recevront des coordonnées matricielles classiques, comme indiqué en Figure 1, à savoir en premier lieu l'indice de ligne `i` (de 0 à 9) et en second lieu l'indice de la colonne `j` (de 0 à 9). Il faudra donc assurer la conversion.

2.3 Vérification d'un coup

Un coup peut être effectué si et seulement s'il s'agit d'un coup autorisé, c'est-à-dire qu'il remplit les conditions suivantes :

- à la position choisie, il se trouve un pion du joueur,
- la destination se trouve sur le damier,
- la destination est vide, c'est-à-dire qu'aucun autre pion ne se trouve à cette position.

La fonction que nous vous demandons d'écrire a le format suivant :

```
checkMove(board, i, j, direction, player)
```

Les paramètres sont les mêmes que pour la fonction `movePiece`, avec en plus un paramètre `player`. La fonction renverra un message d'erreur pour les trois cas décrits et `True` si aucun problème n'est détecté.

2.4 Détection blocage

Avant chaque déplacement il faut vérifier s'il reste encore des mouvements admissibles. Si un joueur ne peut plus déplacer aucun de ses pions, il a perdu la partie ; si aucun de deux joueurs ne peut bouger au moins un pion, il s'agit d'un match nul. La fonction

```
checkEndOfGame(board,player)
```

doit vérifier si la partie est terminée et dans ce cas renvoyer le joueur (-1 ou 1) qui a gagné, 0 pour un match nul et `False` si la partie n'est pas encore terminée. Notez que pour vérifier qu'un variable `a` vaut `0` ou `False`, il faudra faire un test du type « `if a is 0` » ou « `if a is False` », et non « `if a == 0` », qui ne permet pas de faire la différence entre un `0` et un `False`.

2.5 Fonction principale

Finalement, vous allez créer une fonction « `main` » dans un fichier dénommé « draughts.py » qui utilisera les fonctions décrites et qui interagira avec l'utilisateur pour simuler une partie de jeu de dames. C'est-à-dire :

1. Créer le damier initial ;
2. Afficher le damier dans la bonne direction ;
3. Demander au premier utilisateur/joueur de jouer un de ses pions en donnant les coordonnées et la direction ;
4. Vérifier si le mouvement est possible ;
5. Éventuellement effectuer ce mouvement ;
6. Répéter cette procédure avec le deuxième joueur ;
7. Arrêter le jeu, s'il n'y a plus de mouvements possibles.

Votre programme doit s'exécuter avec le commande

```
python3 draughts.py
```

2.6 Fichier de configuration

Le fichier `config.py` contient les variables globales que vous allez utiliser. Ce fichier est fourni avec les énoncés. Vous ne pourrez pas modifier ce fichier car nous l'utiliserons pour tester votre code. Vous devrez comprendre son utilité, et l'utiliser intelligemment, c'est-à-dire vous servir des valeurs qui y sont définie partout où cela se justifie.

2.7 Fichiers à soumettre

Toutes ces fonctions décrites et éventuellement vos fonctions supplémentaires doivent se trouver dans un fichier nommé `draughtsFunctions.py`. Vous allez soumettre deux fichiers :

- `draughts.py`,
- `draughtsFunctions.py`.

2.8 Tester votre projet

Vos projets seront testés en utilisant les fichiers `runTests.py` et `testCases.py` fournis avec les énoncés :

```
python3 runTests.py
```

Avant de soumettre vos projets, il faudra vérifier qu'ils s'exécutent avec le fichier de test. Nous vous conseillons de commencer à travailler sur votre propre project par la création du fichier contenant uniquement les prototypes des fonctions demandées, `draughtsFunctions.py`. Après que vous vous soyez assuré que votre programme (presque vide) s'exécute correctement, vous pouvez commencer à remplir les fonctions une par une.

2.9 Validation et cotation de votre projet

Un projet dont les noms de fonction sont différents que décrit au-dessus ou avec des paramètres différents vaut zéro sans exception ! Par ailleurs, la même règle s'applique pour un projet qui ne peut pas être exécuté avec le script fourni avec les énoncés.

2.10 Consignes

Personne de contact : Catharina Olsen
Remise : Lundi 19 novembre 2012 à 13 heures.
À remettre : Les scripts Python `draughts.py` et `draughtsFunctions.py`

- Au secrétariat étudiants : une version imprimée de ces scripts ;
- Sur l'UV (`http://uv.ulb.ac.be`) : une version électronique de ces mêmes scripts.

3 Partie 2 : On rajoute les règles

Dans cette partie, nous allons ajouter les règles du jeu de dames dont entre autre, celles qui concernent la dame et les prises de pions. Suite à ces ajouts, il va falloir modifier des fonctions de la partie 1. De plus, vous allez devoir implémenter de nouvelles fonctions pour gérer les rafles et les sauvegardes.

3.1 La dame

Lors de la première partie, il vous a été demandé de représenter le damier à l'aide d'un tableau composé des valeurs -1, 0 et 1. Nous adaptons cette structure pour y inclure les dames. Les cases contenant des dames vont garder le même signe mais la valeur absolue sera plus grande que 1. Par exemple dans la Figure 4, le joueur noir possède une dame (valeur -2) sur la ligne 10.

0	2	0	-1	0	-1	0	-1	0	-1
-1	0	0	0	-1	0	-1	0	-1	0
0	-1	0	0	0	-1	0	-1	0	-1
-1	0	-1	0	0	0	0	0	-1	0
0	-1	0	0	0	-1	0	0	0	1
-1	0	0	0	1	0	0	0	1	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	0	0	1
1	0	1	0	1	0	1	0	-2	0

FIGURE 4 – Exemple de représentation interne d'une partie de jeu de dames où chaque joueur a acquis une dame.

La fonction `printBoard` doit également être adaptée pour afficher les caractères « ◎ » (code UTF-8 « 0*x*25*CE* ») et « ⊙ » (code UTF-8 « 0*x*25*C*9 ») qui correspondent respectivement aux dames du joueur noir et du joueur blanc comme sur la Figure 5.

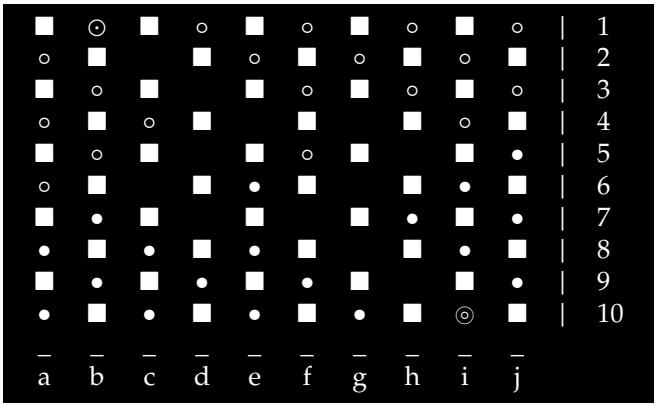

FIGURE 5 – Exemple d'affichage d'une partie de jeu de dames où chaque joueur acquière une dame.

3.2 La prise d'un pion

De même que pour la fonction `printBoard`, nous allons modifier la fonction `movePiece` qui traitera également la prise arrière. Rappelons toutefois que le mouvement (direction) est un mouvement autorisé puisqu'il a été vérifié avant l'appel à `movePiece`.

```
movePiece(board,i,j,direction,length=1)
```

prend les mêmes paramètres qu'avant mais la valeur de *direction* peut être suivie d'un `'B'` qui signifie « arrière ». Exemple :

```
movePiece(board,9,9,'RB')
```

pour une déplacement arrière-gauche.

Cette fois, `movePiece` renverra deux couples `(destI,destJ)` et `(captI,captJ)` au format `((destI, destJ), (captI, captJ))`. Le premier couple correspond aux coordonnées de la case de destination de la pièce qui a bougé. Quant au second couple, il prend une valeur différente de `None` si et seulement si la pièce déplacée saute au-dessus d'un adversaire. Par exemple, dans le cas d'un déplacement simple, la valeur sera `((0,0),None)` et dans le cas d'une prise `((0,0),(1,1))` (capture). **Attention** à respecter l'ordre des coordonnées du format `(i,j)` qui contiennent l'ordonnée et l'abscisse des pièces.

De plus, un paramètre optionnel est ajouté : `length`. Ce paramètre spécifie la longueur (nombre de cases en diagonale) du déplacement et **doit** valoir 1 pour les pions. Le saut au dessus d'une case adverse ne compte que pour un déplacement de longueur 1. Par contre, pour les dames, cette valeur peut varier puisqu'elles peuvent se déplacer de plusieurs cases en une fois. À titre d'exemple, le paramètre `length` va valoir 3 dans ces situations :

– la dame veut avancer de trois cases libres ;
– la dame veut avancer de deux cases libres et sauter au dessus d'une unique pièce adverse ;

mais toujours dans la même direction (diagonale).

3.3 Transformation en dame

Lorsqu'un simple pion atteint la rangée la plus éloignée du joueur, ce pion devient une dame (*king* dans la version anglaise). Dans le cas où le joueur entame une rafle par pion et que celui-ci atteint la rangée opposée après l'une des captures, la rafle se poursuit en tant que pion. La transformation ne s'effectue que si le pion se trouve sur cette rangée après son dernier mouvement. Pour cela, il vous est demandé d'écrire une fonction

```
becomeKing(board,i,j)
```

qui transforme le pion du joueur se trouvant en `board[i][j]` en dame si nécessaire.

3.4 La capture

Vous devez écrire une simple fonction qui enlève la pièce capturée du damier :

```
capture(board,i,j)
```

prend comme paramètre le damier, l'ordonnée et l'abscisse. Par exemple `capture(board,1,1)`.

3.5 La vérification

L'étape de vérification est la plus importante. En effet, le but de cette vérification est de contrôler les déplacements des pions et dames afin d'examiner si le coup est autorisé. Distinguons bien les mots *mouvement* et *coup*. Un *mouvement* est un déplacement d'une pièce dans une direction tandis qu'un *coup* se compose d'un ou plusieurs mouvements pour une même pièce. Donc, à chaque tour, un joueur ne peut effectuer qu'un seul coup.

Un mouvement est autorisé si une pièce se déplace :

– en diagonale ;
– uniquement sur le damier ;
– saute juste au dessus d'une unique autre pièce adverse ;
– en rafle (plusieurs mouvements), la pièce ne peut continuer que tant qu'elle peut capturer (pas d'étape sans capture).
– sur une case libre uniquement vers l'avant pour un pion et dans toutes les directions pour une dame ;
– le longueur du déplacement (nombre de cases en diagonale) pour atteindre une case peut être plus grand que 1 seulement lors d'un mouvement de dame ;

Afin de vérifier cela, il vous est demandé d'écrire deux fonctions dont la première :

```
checkMove(board, i, j, direction, player, length=1, hasPlayed=False,
                                                    hasCaptured=False)
```

Cette fonction est la même que pour la partie 1 du projet sauf que le cas des dames et des prises doit être ajouté. C'est-à-dire, un déplacement arrière sur une case libre et le nombre de mouvements consécutifs dans cette direction (voir le paramètre `length` de `movePiece`). De plus, les paramètres `hasPlayed` (qui vaut `False` au début de chaque tour) et `hasCaptured` (qui vaut `True` si un pion a été pris lors du mouvement précédent de la même rafle) permettent de gérer les rafles. En effet, celui-ci est initialisé à `False` et peut devenir `True` si le joueur a déjà capturé une pièce lors de son tour. Dans ce cas, seul les mouvements de prises (capture d'une pièce adverse) sont autorisés. La valeur de retour est également changée et devient l'un des codes d'erreur décrits en Table 1.

Code d'erreur	Situation
NO_ERROR	Pas d'erreur, le coup est autorisé
PAWN_ONLY_ONE_MOVE	Lorsque le joueur essaie de déplacer le pion de plus d'une case en diagonale
BAD_DIRECTION_FORMAT	Lorsque le format de déplacement n'est pas bon (seulement L, R, LB et RB)
ONLY_KING_GO_BACK	Lorsque le joueur essaie de déplacer le pion en arrière sans capture
SPACE_OCCUPIED	Lorsque le joueur essaie de déplacer une pièce sur une case qui est déjà occupée
CANNOT_JUMP_OUTSIDE	Lorsque le joueur essaie de déplacer une pièce pour capturer un pion mais que la capture aboutit à l'extérieur du damier
TOO_LONG_JUMP	Lorsque le joueur essaie de déplacer une pièce au-dessus de plusieurs autres pièces
CANNOT_GO_OUTSIDE	Lorsque le joueur essaie de déplacer une pièce à l'extérieur du damier
NO_FREE_WAY	Lorsque le joueur essaie de déplacer une dame en capturant une pièce adverse mais veut déposer cette dame ailleurs que dans la case qui suit la capture
NO_PIECE	Lorsque le joueur rentre des coordonnées du damier et qu'à cette position, il n'y a pas de pièce
OPPONENT_PIECE	Lorsque le joueur rentre des coordonnées du damier et qu'à cette position, il y a une pièce adverse
MUST_CAPTURE	Lorsque le joueur poursuit une rafle sans capturer une pièce adverse

TABLE 1 – Codes d'erreur

Notez que vous devrez convertir ce code en un message console plus explicite. Il est recommandé d'implémenter une fonction `strerr(errCode)` qui effectuera la transformation.

Il vous est également demandé d'implémenter une fonction qui peut vous aider dans l'élaboration des autres fonctions :

```
countFree(board,i,j,direction,player=None,length=0)
```

Celle-ci prend en paramètre le damier *board*, la position du joueur `i,j` ainsi que sa direction `direction` ('L', 'R', 'LB' ou 'RB') dans le but de détecter le longueur de mouvements à effectuer avant de capturer une pièce adverse. Le paramètre `player` sert à identifier le joueur 1 ou −1. Si sa valeur n'est pas spécifiée (`None`), il faut la mettre à jour avec le contenu de `board`. Enfin, le paramètre `length` est le nombre de cases vides déjà vérifiées par appels récursifs depuis la position `(i,j)` dans la direction donnée.

3.6 Fin de partie

Lors de la partie 1 du projet, il vous a été demandé de développer une fonction `checkEndOfGame` qui vérifie si les joueurs peuvent encore bouger des pièces. Il est possible que les joueurs arrivent à une situation où aucun des deux ne peut vaincre l'autre, par exemple chacun a une dame. C'est pourquoi, lors de l'interaction avec les joueurs (lorsqu'ils choisissent leur pièce à déplacer), le choix « Partie nulle » doit être proposé. Lorsqu'un joueur propose ce choix, la demande est faite à l'adversaire. S'il accepte, la partie s'arrête par match nul, sinon la partie reprend et le joueur qui a fait la demande continue la partie. **Remarque :** il ne vous est pas demandé de détecter les cas des parties « sans fin ».

Vous devez également adapter la fonction `checkEndOfGame`. Cette fonction renvoie le joueur victorieux (1 ou −1), 0 en cas de match nul ou `False` si la partie n'est pas terminée. Un joueur est victorieux lorsque son adversaire n'a plus de pièce ou lorsque son adversaire est bloqué. Un joueur est bloqué lorsqu'il a le trait et qu'il ne peut plus bouger au moins l'une de ses pièces. Le cas du match nul se produit quand les deux joueurs sont bloqués. C'est-à-dire lorsque les deux joueurs ne peuvent plus bouger au moins une de leurs pièces.

3.7 Fonction principale

Après avoir implémenté et testé votre code, il vous faut prendre en compte les changements et nouvelles fonctions dans la boucle principale de votre programme. Pour ce faire, votre fichier draught.py, devra se composer des étapes suivantes :

1. Initialisation du damier ;
2. Si `checkEndOfGame` renvoie un code de fin de partie, aller à l'étape 11 ;
3. Proposer au joueur de déplacer une pièce (coordonnées), de sauvegarder/restaurer la partie (et le faire si nécessaire), ou de proposer une partie nulle à l'adversaire ;
4. Demander au joueur le coup à effectuer (direction et le nombre de cases consécutives *length* dans cette direction) ou s'il a fini son tour ;
5. Si le joueur a terminé son tour, appeler la fonction `becomeKing` et poursuivre à l'étape 10, sinon continuer ;
6. Si le coup n'est pas permis par `checkMove`, continuer à l'étape 4 ;
7. Déplacer la pièce à l'aide de `movePiece` ;
8. Capturer une pièce si nécessaire ;
9. Continuer à l'étape 4 ;
10. Passer au joueur suivant, retourner à l'étape 2 ;
11. Fin de partie, afficher le résultat.

3.8 Sauvegarde et restauration de partie

Nous vous demandons d'ajouter deux fonctions qui correspondent respectivement à la sauvegarde et à la restauration d'une partie :

```
save(filename,board,player)
load(filename)
```

Nous ne vous demandons pas d'optimiser la structure des fichiers mais de respecter le format défini ci-après. Les fichiers sont au format texte et leur nom se termine par « .dat ». La première ligne contient le joueur qui a le trait (−1 ou 1) et la seconde ligne contient la taille du damier (*dimension*). Les lignes suivantes correspondent aux valeurs du damier séparées par une ou des espaces. Par exemple :

```
-1
10
 0  2  0 -1  0 -1  0 -1  0 -1
-1  0  0  0 -1  0 -1  0 -1  0
```

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

Remarque : Le retour à la ligne peut être spécifié par le caractère « `\n` » en python.

3.9 Fichiers

Comme pour la première partie, vous avez reçu un fichier « config.py » et un fichier de tests unitaires. Les fichiers sources à soumettre sont **uniquement** « draught.py » et « draughtsFunctions.py ». Le fichier « draughtsFunctions.py » doit contenir toutes les fonctions demandées et doit être fonctionnel avec votre « draught.py ».

3.10 Rapport

En plus du code, on demande un rapport de 4 à 5 pages (page de garde et table des matières non comprises). Plus en détails, ce rapport doit comprendre :

- Une page de garde qui reprend vos coordonnées, la date, le titre, etc. ;
- La table des matières ;
- Une introduction et une conclusion ;
- Des sections, comprenant ce qui a été fait dans le projet jusqu'à ce point (la partie 1 comprise) ;
- Des exemples des fichiers de résultat ;
- Des références si nécessaire.

Le rapport doit détailler les difficultés rencontrées, l'analyse et les solutions proposées, ainsi qu'une bonne explication des algorithmes (pseudo-code avec des exemples, et pas le code source). Il doit être clair et compréhensible pour un étudiant imaginaire qui aurait suivi le cours INFO-F101 (Programmation) mais n'aurait pas fait le projet, sans trop d'informations ni trop peu. Toutes les étapes doivent être détaillées et présentées dans une séquence logique. Expliquez toutes les notions et les terminologies que vous introduisez.

Le rapport peut être écrit avec des logiciels de traitement de texte (uniquement Microsoft Office, OpenOffice ou LibreOffice), mais il est obligatoire d'utiliser les outils de gestion automatique des styles, de la table de matière et de numérotation de sections. Nous vous conseillons cependant d'utiliser le système LaTeX, très puissant pour une mise en page de qualité.

On rappelle qu'une bonne orthographe sera exigée.

En annexe, un document donne des conseils sur la rédaction d'un bon rapport.

3.11 Consignes

Personne de contact : Jean-Sébastien Lerat
Remise : Lundi 17 décembre 2012 à 13 heures.
À remettre : Les scripts Python `draughts.py` et `draughtsFunctions.py` ainsi que le rapport :

- Au secrétariat étudiants : une version imprimée de ces scripts et du rapport ;
- Sur l'UV (`http://uv.ulb.ac.be`) : une version électronique de ces mêmes scripts et du rapport, dans son format source (.doc(x), .odt, ou .tex) et sa version PDF.

4 Partie 3 : Interface graphique

La troisième partie du projet consiste en la mise en œuvre d'une *interface graphique* (ou GUI, acronyme anglais pour « *graphical user interface* ») pour votre logiciel qui permettra à l'utilisateur de jouer au jeu de dames à l'aide de la souris. Python fournit par défaut une bibliothèque nommée *Tkinter* dans ce but. D'autres bibliothèques graphiques pour Python existent également, telles PyQt, PyGTK ou encore wxPython, mais nous exigeons que vous utilisiez Tkinter dans le cadre de ce projet dans un souci de faciliter le travail des correcteurs. Par ailleurs, nous exigeons qu'au terme de cette troisième partie, votre code gère les erreurs à l'exécution à l'aide de gestion d'exceptions telles que vues au cours de programmation.

4.1 Structure et fonctionnement d'une GUI

Une GUI est typiquement composée d'un ensemble d'éléments nommés *widgets* agencés ensemble. Une fenêtre, un bouton, une liste déroulante, une boite de dialogue ou encore une barre de menus sont des exemples de widgets typiques. Il peut aussi s'agir d'un *conteneur* qui a pour rôle d'agencer d'autres widgets en son sein.

Une GUI va typiquement être structurée, de manière interne, sous forme d'un *arbre* de widgets. Dans Tkinter, tout programme doit avoir un widget « racine » qui est une instance de la classe *tkinter.Tk*, créant en fait une fenêtre principale, dans laquelle on peut ensuite ajouter d'autres widgets. À l'appel d'un constructeur de widget, il vous faudra toujours préciser en paramètre quel sera son widget parent (exception faite du widget racine, bien sûr).

La structure visuelle d'une GUI (aussi appelée *maquette*) n'est que la moitié du travail ; il faut que votre code des deux parties précédentes puisse être lié à cette interface. À l'exécution, une GUI va réagir à des *évènements* comme un clic sur un bouton. Votre programme ne va donc plus simplement se lancer, faire des opérations, puis se terminer ; l'interface va faire une boucle d'attente d'événements. Tkinter offre notamment la possibilité de lier le clic sur un bouton à l'appel d'une fonction donnée (voir le paramètre `command` du constructeur de la classe *tkinter.Button* par exemple) ; il vous faudra savamment utiliser ces possibilités pour mettre à jour l'affichage au besoin.

4.2 Exemple de maquette et contraintes

La Figure 6 vous donne un exemple de maquette pour l'interface graphique. Nous vous demandons de respecter celle-ci dans la mesure du possible. Nous vous imposons notamment les éléments suivants :

1. Une zone affichant le damier avec tous les pions ;
2. Deux zones indiquant aux joueurs le nombre de pions adverses qu'ils ont pris ;
3. Une zone de statut indiquant quel joueur a le trait ;
4. 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.

Il doit être possible de fermer votre programme en cliquant sur le bouton de fermeture de la fenêtre. Notez qu'il ne vous est pas imposé de réaliser l'interface en anglais comme à la Figure 6. Le choix des couleurs, la taille des cases, des pions et les fontes de caractères ainsi que les différences de représentation entre les pions simples et les dames ne sont pas imposés.

Remarque sur la performance. Si vous liez une fonction qui nécessite un temps certain de calcul à un bouton de votre interface par exemple, votre interface restera « gelée » à chaque clic jusqu'à ce que la fonction termine son exécution, ce qui n'est pas vraiment idéal. Dans le cadre de ce projet, nous nous contenterons de ce comportement et ne le pénaliserons pas. Pour le résoudre, il faut exploiter le concept de *multithreading* qui sort du cadre de ce projet (et du cursus de BA1 par ailleurs) et qui sera exploré notamment aux cours INFO-F201 (Systèmes d'exploitation) et INFO-F202 (Langages de programmation 2) en BA2.

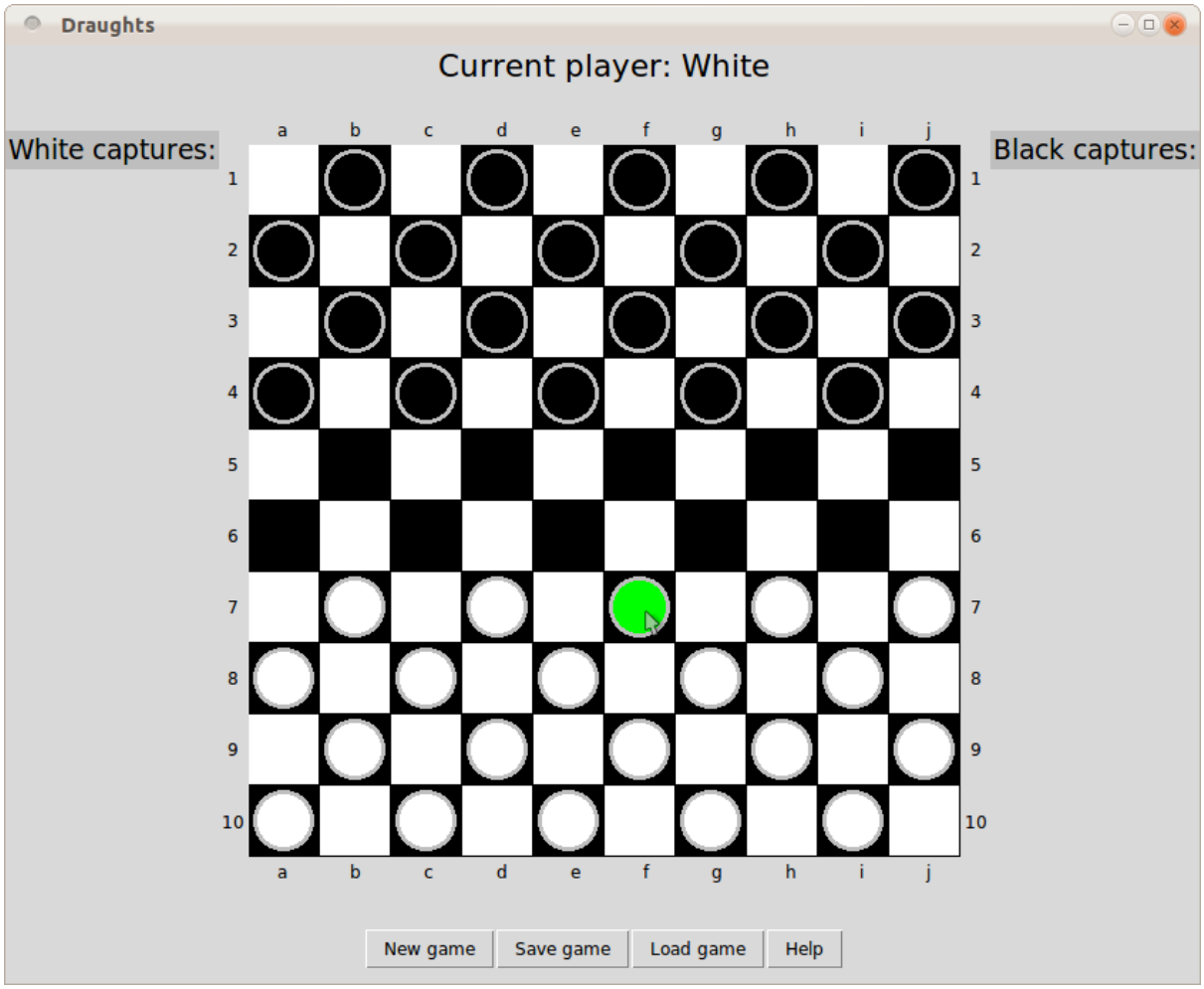

FIGURE 6 – Exemple de maquette *tkinter*.

4.3 Références utiles

Voici une liste (non exhaustive, bien entendu) de documents, disponibles gratuitement en ligne, pouvant vous aider à développer votre interface graphique avec tkinter :

- `http://www.xavierdupre.fr/mywiki/InitiationPython`
- `http://wiki.python.org/moin/TkInter`
- `http://www.pythonware.com/library/tkinter/introduction/` (en anglais pour python 2 !)

4.4 Comportement de GUI

Votre interface graphique 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 re-cliquer sur le même pion pour 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 (le pion 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 (e.g. *"le coup e5-f3 n'est pas permis"* ou *"impossible de sauvegarder la partie"*) ;
- Si le joueur clique sur autre chose qu'un de ses pions le programme ne doit rien faire. Autrement dit il ne faut pas afficher un message d'erreur si le jour fait un clic sur une case vide (avant de cliquer sur un de ses pions).

Notez que en récupérant les coordonnées de clics, l'interface peut facilement se rendre compte que l'utilisateur veut faire bouger un pion, par exemple de la case $(f,7)$ à la case $(g,6)$. Mais ce n'est pas le format requis par les fonctions écrites lors des parties précédentes. L'interface devra donc convertir cette demande en, par exemple, `checkMove(board, 6, 5, 'R', 1, 1)`. Il se peut cependant que la conversion ne soit pas possible : il est impossible par exemple de convertir le coup $(e,5)-(f,3)$ puisqu'il ne s'agit pas d'un déplacement en diagonale. Il faudra donc afficher le message d'erreur adéquat.

4.5 Dessin sur un canevas Tkinter

Nous vous fournissons ici un algorithme qui vous permettra de dessiner des rectangles et des cercles dans un canevas Tkinter (c'est-à-dire une instance de la classe *tkinter.Canvas*). En effet, nous ne souhaitons pas que vous génériez une image avec le plateau qui soit ensuite chargée, nous voulons que vous utilisiez Tkinter directement pour afficher les carrés du plateau de jeu et les pions.

```
boardFrame = Frame(masterFrame) # création de la Frame
boardCanvas = Canvas(boardFrame, height=500, width=500) # définit la taille du canevas

# Dessine un rectangle noir de taille 100x100
# Le coin supérieur gauche se trouve en coordonnées (50, 80)
boardCanvas.create_rectangle(50, 80, 150, 180, fill="black")

# Dessine un cercle (rouge avec un contour vert) inscrit dans le carré
boardCanvas.create_oval(50,80,150,180, outline="green", fill="red", width=2)

# Place la frame avec le plateau dans le canevas
boardFrame.grid(row=0, column=0)

# On peut aussi utiliser "pack()":
#boardFrame.pack()
```

Remarquez que les rectangles sont définis à l'aide de deux points : le coin supérieur gauche et le coin inférieur droit. Les ovales (ainsi que les cercles et les ellipses) sont définis à l'aide des rectangles (l'ovale est alors inscrit dans un rectangle).

Notez que ce code ne fonctionnera bien entendu pas par magie ; il vous reviendra de trouver où placer cet algorithme pour qu'il fasse correctement fonctionner votre interface.

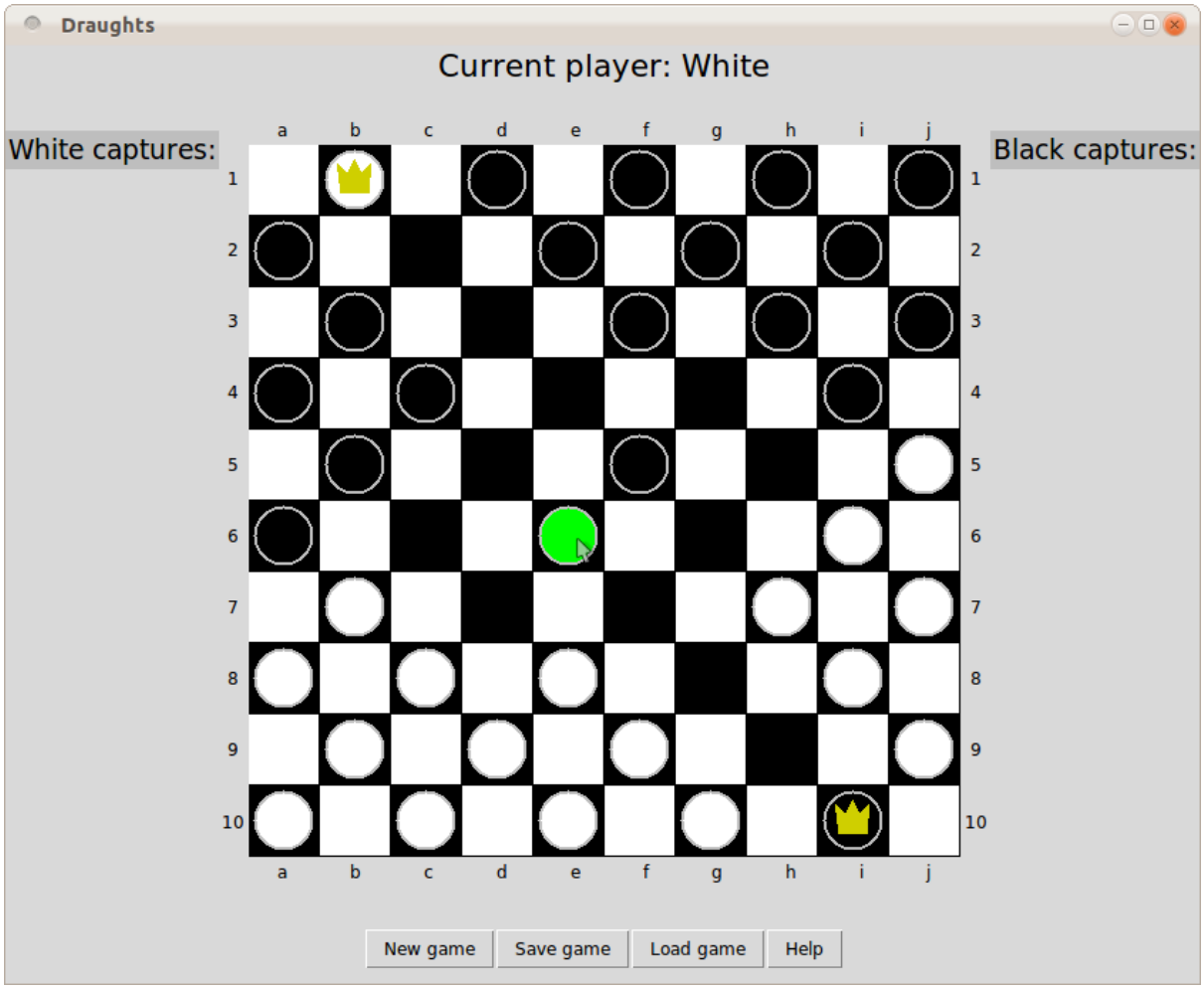

FIGURE 7 – Exemple de maquette *tkinter*.

4.6 Gestion d'exceptions

Nous vous demandons également dans cette phase d'implémenter une gestion d'exceptions dans votre logiciel. Nous nous attendrons donc à ce que vous utilisiez pleinement et intelligemment les constructions `try-except-finally` ainsi que la commande `raise` pour gérer proprement les erreurs pouvant survenir lors de l'exécution de votre programme. En particulier, nous nous attendons à ce que les erreurs relatives à des données erronées en entrée donnent lieu à l'affichage de boîtes de dialogue décrivant ce qu'il s'est passé (utilisez pour cela le module *messagebox* fourni avec Tkinter). Vous trouverez un exemple de telle boite de dialogue en Figure 8.

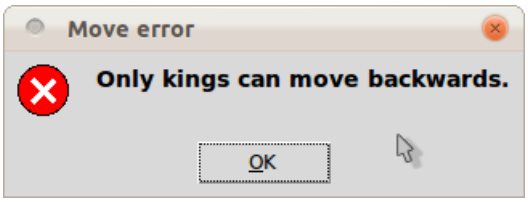

FIGURE 8 – Boîte de dialogue d'erreur avec *messagebox.showerror("Move error", "Only kings can move backwards.")*.

4.7 Utilisation de classes

Dans la mesure du possible, nous souhaitons que vous fassiez un usage judicieux de classes (et donc exploiter constructeurs, attributs et méthodes) pour structurer votre code.

4.8 Exécution du programme

Nous demandons que votre interface graphique soit lancée via un autre programme, nommé `draughtsGUI.py`, qui ne prendra bien entendu pas de paramètres en ligne de commande puisque l'interface-même sert à les fournir. Votre programme `draughts.py` doit par contre toujours fonctionner. Pensez à modulariser votre code de manière appropriée ; nous ne souhaitons pas voir du code copié-collé dans les programmes `draughts.py` et `draughtsGUI.py` !

4.9 Consignes

Personne de contact : Nikita Veschikov
Remise : Lundi 25 février 2013 à 13 heures.
À remettre : Les scripts Python `draughts.py`, `draughtsFunctions.py` et `draughtsGUI.py` :
– Au secrétariat étudiants : une version imprimée de vos scripts Python ;
– Sur l'UV (`http://uv.ulb.ac.be`) : une version électronique de ces mêmes scripts.

5 Partie 4 : Humain vs Machine

La quatrième et dernière partie de ce projet consiste à mettre en place une "*intelligence artificielle*" (IA) permettant à un joueur humain de jouer contre un ordinateur ayant un faible niveau. Pour cela, l'implémentation d'une stratégie vous est demandée. Cette stratégie est une succession d'étapes à réaliser par l'ordinateur lorsqu'il a le trait. Par nature, cette stratégie vous permettra d'effectuer plusieurs jeux contre un ordinateur ayant des choix différents quand bien même votre stratégie est constante.

La première étape exécutée par l'intelligence artificielle est la vérification par l'ordinateur qu'une prise d'un ou de plusieurs pions adverses est réalisable. Cette vérification s'effectue en testant le déplacement d'un parmi N de ses pions, choisis uniformément aléatoirement, où N est une variable définie dans le fichier config.py. Par défaut la valeur de N est 10. Dans le cas où une prise est détectée, l'ordinateur l'effectue. À la suite d'une prise simple, l'IA vérifie si une autre prise, avec le même pion, est possible afin de réaliser une prise multiple. En d'autres mots, l'IA devra faire autant de prises que possible avec le pion choisi.

Dans le cas où aucune prise n'a été détectée, l'ordinateur vérifie si un pion, choisi au hasard en testant N de ses pions, est déplaçable sans être mangé par l'adversaire au moyen d'une prise simple par un pion ou une dame[3]. Dans le cas où ce pion est trouvé, l'ordinateur réalisera ce déplacement, autrement l'ordinateur déplacera un de ses pions choisi aléatoirement.

Du point de vue technique, il est nécessaire de détenir l'ensemble des coordonnées des pions de l'ordinateur afin de pouvoir en choisir un au cours de l'exécution de la stratégie. Pour cela, une liste de tuples regroupant l'ensemble des coordonnées de ses pions devra être stockée en mémoire. Chaque tuple contiendra les deux coordonnées permettant de retrouver un pion sur le damier. Au début de la partie ou lors du chargement d'une partie, l'ordinateur devra créer cette liste. Pour cela, un parcours du damier sera effectué afin de déterminer la position de chaque pion de l'ordinateur. Pendant le jeu, dès que l'ordinateur a le trait, une vérification s'effectuera sur cette liste afin de s'assurer qu'elle contient l'ensemble des positions des pions de l'ordinateur. Pour cela, un parcours du damier en fonction des positions enregistrées dans la liste sera effectué. Dans le cas où une position enregistrée ne contient pas un pion de l'ordinateur, cette position est supprimée de la liste.

Un résumé de l'ensemble de la stratégie est disponible dans l'Algorithme 1.

Algorithme 1 Résumé de l'intelligence artificielle

```
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 pion de l'ordinateur choisi de manière aléatoire
end if
```

5.1 Rapport

En plus du code, on demande un rapport de 9 à 10 pages (page de garde et table des matières non comprises), avec au minimum 4000 mots. Comme pour le premier, ce rapport doit comprendre :

- Une page de garde qui reprend vos coordonnées, la date, le titre, etc. ;
- La table des matières ;
- Une introduction et une conclusion ;
- Des sections, comprenant ce qui a été fait dans le projet, donc les parties 1, 2, 3 et 4 ;
- Des références si nécessaire.

Il peut aussi comprendre des annexes (facultatif) si vous pensez qu'elles sont nécessaires pour des explications supplémentaires (comme des parties de code). Les annexes ne sont pas toujours lues, donc

3. Les prises multiples ne sont donc pas considérées.

les informations indispensables à la compréhension du projet ne peuvent pas s'y trouver. Les annexes ne comptent pas non plus dans les 9–10 pages de rapport.

Les instructions données pour le premier rapport s'appliquent également ici.

Vous pouvez reprendre des parties du premier rapport, mais faites attention à la continuité de l'ensemble du rapport. Il ne suffit pas de rajouter les parties 3 et 4 au document précédent, il faut retravailler le rapport dans son ensemble. Si dans les parties 1 et 2 il y avait des erreurs de contenu ou autre (orthographe, typographie. . .), elles doivent être corrigées.

Dans l'introduction et la conclusion, tout le projet doit être présenté avec ses objectifs et résultats.

Dans les sections, tout ce que vous avez fait doit être expliqué. Vous devez décrire le projet dans son ensemble, et non partie par partie. Ainsi, évitez les noms de section tels que « Partie 1 », « Partie 2 », etc.

On rappelle que le rapport doit être compréhensible par tout étudiant (imaginaire) qui aurait suivi le cours INFO-F101 (Programmation) mais n'aurait pas fait le projet.

Pour des conseils sur la rédaction d'un bon rapport, nous vous renvoyons à nouveau vers le document en annexe.

5.2 Présentation

À coté d'un rapport écrit qui explique un projet en détails, il est souvent demandé, dans les travaux scientifiques, une petite présentation orale qui résume le projet. On vous demande donc une présentation de 10 minutes (pas plus !), où, à l'aide de transparents, vous nous expliquez votre projet.

La présentation ne doit pas aller dans les détails, mais expliquer l'idée générale et les objectifs du projet, les méthodes utilisées et les résultats obtenus. Des exemples et des figures aident beaucoup dans un exposé oral.

Pour des conseils sur la réalisation d'une bonne présentation, voir de nouveau le document en annexe. On remarque juste que pour un exposé de 10 minutes, il faudrait compter maximum 10–12 transparents (la règle empirique est 1 transparent par minute).

Pour vous aider à préparer votre présentation, voici quelques éléments que nous voudrons y trouver :

1. Structure de l'exposé :
   - Une introduction de la présentation (ce dont on va parler) ;
   - Une introduction du projet (les objectifs) ;
   - Différentes sections clairement identifiables, expliquant la contribution, les difficultés rencontrées, les solutions apportées, éventuellement les problèmes non résolus, etc. ;
   - Une conclusion (ce que l'auditoire doit retenir).
2. Qualité du support :
   - Des transparents pas trop chargés, sans code source. Éviter les détails de trop bas niveau (nom/signature de fonctions, de variables. . .) ;
   - Des illustrations, des schémas, des diagrammes (découpe du programme, lien entre les parties, schémas techniques. . .) ;
   - Une orthographe soignée, un vocabulaire adéquat, une uniformité d'aspect entre les pages ;
   - Des transparents numérotés.
3. Qualité de l'exposé :
   - Respect des contraintes temporelles ;
   - Pas trop d'hésitation, pas devoir chercher ses mots, mais pas non plus réciter un texte par cœur ;
   - Présence, volume de la voie, articulation, regarder l'auditoire...

Le calendrier des démos et défenses apparaîtra en mars. Les démos auront lieu pendant la semaine de cours 19, les défenses pendant la semaine de cours 20.

5.3 Consignes

Personne de contact : Liran Lerman

Remise : Lundi 25 mars 2013 à 13 heures.

À remettre : Les scripts Python `draughts.py`, `draughtsFunctions.py`, `draughtsGUI.py`, ainsi que le rapport :

- Au secrétariat étudiants : une version imprimée de ces scripts, et du rapport final ;

– Sur l'UV (`http://uv.ulb.ac.be`) : une version électronique de ces mêmes scripts et du rapport, dans son format source (.doc(x), .odt, ou .tex) et sa version PDF.