Documentation développeur

1. Introduction

J'ai décidé de coder le jeu des dames chinoises. On peut jouer avec deux modes différents : Le mode classique et le mode rapide (fast-paced) qui est utilisé en France principalement.

Cette documentation développeur donne toutes les informations nécessaires pour comprendre la structure du code, la localisation et utilité de certaines fonctions, ainsi que le fonctionnement de certains algorithmes un peu plus complexes.

2. Structure du code

Le programme est découpé en 10 fichiers python afin d'améliorer la lisibilité du code : ChineseCheckers.py, MainMenu.py, AboutandHowtoPlay.py, Setup.py, Game.py, GameOver.py, GraphicModeFunctions.py, GameState.py, AI.py et ManageSaves.py. Il y a également un fichier icon.ico qui est l'icône du jeu que l'on retrouve en haut à gauche des fenêtres. Détaillons les fonctionnalités de chaque fichier.

2.1. ChineseCheckers.py

Le fichier ChineseCheckers.py contient la fonction main qui va lancer tout le programme. C'est donc ce programme qu'il faut exécuter dans le terminal pour jouer.

2.2. Les fichiers du GraphicMode

Le code contient 5 fichiers qui se chargent de faire la partie graphique du jeu. Chaque fichier contient une classe qui correspond à une fenêtre du jeu (2 classes pour le fichier AboutandHowtoPlay.py). Afin d'implémenter la partie graphique, j'ai utilisé wxPython, qui est une bibliothèque utilisée pour la création d'interfaces graphiques.

Le jeu peut afficher 6 fenêtres différentes. Dans les 5 fichiers se répartissent donc 6 classes différentes :

- MainMenu
- HowToPlay
- About
- Setup
- Game
- GameOver

Les 6 classes sont très similaires. Elles créent toutes un « Frame » de taille (1000, 700) fixe et affiche l'icône du jeu que j'ai créée. Chaque classe (sauf MainMenu) prend un paramètre « previous ». Elles possèdent toutes une fonction « drawScreen » qui affiche l'écran dont il est question avec tous ses boutons, textes, etc… Toutes les classes sont liées à l'évènement wx.EVT_PAINT qui dessine tous les cercles et rectangles à chaque fois que la fenêtre doit être redessinée (pendant le jeu par exemple),

et à l'évènement wx.EVT_CLOSE, qui généralement détruit la fenêtre actuelle et ouvre la fenêtre précédente (qui est toujours la fenêtre MainMenu) lorsque l'utilisateur ferme la fenêtre.

Les différents boutons présents sur les fenêtres sont reliés à des fonctions (grâce à des évènements wx)qui permettent de décider ce qui doit être fait. Les boutons radio et toggle ont des caractéristiques spéciales qui sont aussi contrôlées dans les fonctions correspondantes.

Voyons maintenant les classes de plus près.

La classe MainMenu affiche le menu principal. A partir de ce menu, on peut ouvrir la fenêtre contenant les règles (HowToPlay) ou celle contenant des informations sur le développement du jeu (About) ou alors on peut lancer la partie en cliquant sur le bouton PLAY. Une fois que ce bouton est appuyé, on peut choisir de lancer une nouvelle partie (bouton « New Game »), ce qui entrainera l'ouverture la fenêtre Setup où les joueurs doivent mettre en place leur partie. Si une partie a déjà été créée au préalable, le bouton « Load Game » sera cliquable et ouvrira directement la fenêtre Game en donnant à cette classe le return de la fonction extractSaveFile située dans le fichier ManageSaves (détails dans la partie 2.6).

Les classes Rules et About affichent seulement du texte donnant des informations, et un bouton « Back » qui permet de revenir au menu principal.

La classe Setup est appelée lorsqu'un nouveau jeu est créé. Elle permet au(x) joueur(s) de choisir le mode de jeu (1 contre 1 ou 1 contre ordinateur), la difficulté du jeu (si jeu contre ordinateur), leurs(son) nom(s), leurs(sa) couleur(s). Lorsque le joueur lance la partie, une classe GameSetup (présente dans le fichier GameState.py, détails dans la partie 2.4) est créée et on y met toutes les informations concernant la configuration du jeu choisie par le joueur.

La classe Game est un peu plus complexe que les autres. En plus de « previous » elle prend en paramètre des classes GameState et GameSetup (détails dans la partie 2.4) Elle peut être appelée de deux façons différentes : soit elle est appelée par la classe Setup quand une nouvelle partie est lancée, soit par la classe MainMenu quand la partie sauvegardée est lancée. Dans le premier cas, la variable self.gameSetup prend comme valeur le paramètre GameSetup. De plus, une nouvelle classe GameState est appelée (se situe dans le fichier GameState.py, détails dans la partie 2.4) et est stockée dans la variable self.gameState. Dans le deuxième cas, les variables self.gameState et self.gameSetup prennent les valeurs de GameState et GameSetup qui sont en paramètre et qui avaient été créée en avec la fonction extractSaveFile (se situe dans le fichier ManageSaves.py, détails dans la partie 2.6). Détaillons maintenant quelques fonctions de cette classe.

La fonction positionToCell prend en paramètre la position de l'endroit où a cliqué le joueur en pixels et convertit ces valeurs en case de la grille. Si le joueur n'a pas cliqué sur un cercle, la fonction renvoie None, sinon, elle renvoie la colonne et ligne dans la grille. La méthode utilisée pour faire la conversion sera détaillée dans la partie 3.2.

La fonction onClick est appelée dès qu'un évènement EVT_LEFT_DOWN a lieu, quand un joueur clique sur l'écran. Cette fonction récupère la position en pixels de l'endroit cliqué puis appelle la fonction positionToCell pour obtenir la cellule de la grille correspondante. Si l'endroit cliqué était bien dans un cercle (la fonction positionToCell ne retourne pas None), alors la fonction play du fichier GameState.py est appelée (voir partie 2.3) et la fenêtre est redessinée avec toutes ses nouvelles valeurs à l'aide de la fonction Refresh. Si la fonction play renvoie « turn is over » ça veut dire que le joueur actuel a fini son tour et on passe au suivant. Afin d'indiquer à qui est le tour, en haut à droite de l'écran, le nom du joueur actuel est coloré en jaune. Enfin, si la fonction GameIsOver (située dans le fichier GameState.py) renvoie True, on donne la bonne valeur à la variable winner de GameState et la fonction onGameOver est appelée.

La fonction onUndo est appelée lorsqu'un joueur clique sur le bouton « Undo ». En se servant des variables previousMove, previousNbSimpleMoves et previousTotalNbJumps contenues dans la variable gameState, cette fonction permet de revenir un mouvement un arrière. Cette fonction ne peut pas être appelée deux fois d'affilée (le bouton « Undo » est donc désactivé dès qu'il est cliqué et réactivé au tour d'après dans la fonction onClick). De plus, ce bouton ne peut pas être appelé tant que le joueur actuel n'a pas fini son tour (s'il a commencé à faire bouger une bille mais pas encore validé le mouvement. Donc au milieu d'un tour également, le bouton « Undo » est désactivé.

Enfin, la classe GameOver affiche une fenêtre qui annonce la fin du jeu. S'affiche alors le nom de celui qui vient de gagner la partie, le score total (le nombre de parties gagné par chacun) et des statistiques sur le jeu (nombre de mouvements, nombre de mouvements simples et sauts). Le joueur peut alors décidé de lancer une nouvelle partie contre la même personne (bouton « Play Again »), de sauvegarder la partie puis revenir au menu principal (bouton « Save & Quit ») ou de quitter le jeu (bouton « Exit »).

2.3. GraphicModeFunctions.py

Le fichier GraphicModeFunctions.py permet de compléter le fichier GraphicMode.py avec des fonctions qui sont très utilisées dans toutes les classes. On évite ainsi un maximum de redondance de code.

On y trouve tout d'abord toutes les variables contenant les couleurs qui sont utilisées dans le programme (les couleurs des différentes billes, de l'arrière-plan, du plateau, etc…).

Il y a ensuite 10 fonctions qui sont de 3 types différents :

- Les fonctions « display » qui affichent directement quelque chose sur la fenêtre quand celles-ci sont appelées. Il y en a 3 : displayButton (qui affiche un bouton et le lie avec la fonction correspondante), displayRadioButtons (qui affiche directement les 6 radioButtons) et displayText (qui affiche le texte voulu au bon endroit et de la bonne taille). Ainsi, dans le fichier GraphicMode.py, seulement une ligne sera nécessaire pour afficher ce que l'on veut. Cependant, en utilisant ces fonctions, le bouton ou texte correspondant n'aura pas de variable et ne pourra donc pas être changé au besoin. C'est pour cela que l'on a parfois besoin d'utiliser la prochain type de fonctions.
- Les fonctions « setup » qui prennent en paramètre une variable et la change d'une certaine façon. Il y en a 3 : setupButton (qui sert notamment pour les toggleButton en leur donnant la bonne taille, police et couleur), setupTextControl (qui donne la bonne taille, position, police et taille maximale à un textControl, qui est un endroit où l'utilisateur peut écrire du texte) et setupTextInCenter (qui prend un texte et le met au centre de l'endroit demandé). Ces fonctions requièrent la création de la variable en amont.
- Les fonctions « draw » qui dessinent quelque chose. Il y en a 4 : drawRectangle (qui dessine un rectangle de la bonne taille, au bon endroit, avec la bonne couleur et bordure), drawOneCircle (qui dessine un cercle d'une certaine couleur et l'endroit demandé et de la taille demandée), drawCircles (qui dessine les 6 cercles de 6 couleurs différentes avec l'espacement demandé), drawLine (qui dessine une ligne horizontale de taille fixe au bon endroit).

2.4. GameState.py

Le fichier GameState.py contient toutes les informations nécessaires pour pouvoir jouer. Il contient deux classes : la classe GameSetup qui contient les informations concernant la configuration du jeu (noms des joueurs, mode de jeu, difficulté de l'IA, etc…) et la classe GameState qui regroupe toutes les informations concernant l'état actuel de la partie (placement des billes) et les règles du jeu.

La classe GameSetup est très courte, puisqu'elle ne contient que quelques variables et aucune fonction.

En revanche, la classe GameState est plus complexe. Elle contient les variables concernant la grille, le score, les statistiques de la partie, etc… Dans l'initialisation de la classe, on appelle la fonction initializeGrid, qui initialise la grille en donnant à chaque cellule sa bonne valeur (None, 0, 1 ou 2). L'algorithme utilisé sera détaillé dans la partie 3.1. Elle contient aussi toutes les fonctions qui permettent de donner au jeu des règles (play, isGameOver, makeListOfAllowedMoves …). Détaillons quelques fonctions.

La fonction play prend en paramètre la cellule cliquée par un joueur. Elle retourne soit « turn is over », soit None. Il y a alors 4 cas possibles :

- Le joueur reclique sur le cercle qui est déjà sélectionné et il a déjà bougé ce cercle durant ce tour : son tour se termine (return « game is over ») et les variables nécessaires sont réinitialisées.
- Le joueur clique sur un cercle qui contient une de ses billes et il n'a pas encore bougé de billes dans ce tour : ce cercle devient donc sélectionné et on appelle la fonction makeListOfAllowedMoves qui complète la variable allowedMoves avec tous les mouvements possibles.
- Le joueur clique sur un cercle vide, une bille a été sélectionnée auparavant et le mouvement de la bille sélectionnée vers le cercle vide est autorisé (on le vérifie avec la fonction moveIsAllowed) : on effectue le mouvement (la cellule contenant la bille sélectionnée prend la valeur 0 et le cercle vide prend la valeur du joueur actuel). Puis, si le mouvement était simple (vers un cercle directement adjacent), la fonction retourne « turn is over ». Si le mouvement était un saut, le joueur peut continuer de jouer à partir de ce cercle et on appelle la fonction makeListOfAllowedMoves.
- Si le tour n'est pas fini, la fonction retourne None.

La fonction makeListOfAllowedMoves prend en paramètre la position dans la grille de la bille qui a été sélectionnée par le joueur et retourne une variable contenant tous les mouvements possibles de cette bille. L'algorithme ne sera pas tout à fait le même selon le mode de jeu choisi. Le mode de jeu classique permet aux joueurs de bouger leur bille soit sur une place vide adjacente, soit de sauter par-dessus une bille qui est adjacente et se placer directement derrière(si la case est vide). Le mode de jeu rapide permet également aux joueurs de bouger leur bille vers une place vide adjacente, mais également de faire un saut par-dessus une bille qui n'est pas forcément directement adjacente.

2.5. AI.py

Le fichier AI.py, qui contient la classe AI, contient toutes les informations et fonctions qui concernent « l'intelligence artificielle ». Cette classe possèdent 3 variables : self.marblesPos (qui contient la position actuelle des billes de l'IA), self.allPossibleMoves (qui contient tous les mouvements qui sont possibles pour l'IA lors d'un tour donné) et self.points (qui à chaque mouvement possible associe un score).

Lorsque le jeu est en route, c'est toujours la fonction moveAI qui est appelée à partir de la classe Game. Cette fonction se charge de créer la liste des mouvements possibles et retourner la mouvement choisi en appelant les fonctions correspondantes.

La fonction makeListOfAllPossibleMoves appelle la fonction makeListOfAllowedMoves de la classe GameState afin de créer une liste contenant tous les mouvements possibles de toutes les billes (même les mouvements composés).

La fonction chooseMove choisi le mouvement qui va être effectué par l'IA en appelant les différentes fonctions qui rajoutent et enlèvent des points aux différents mouvements possibles. Le choix de ce mouvement dépend de la difficulté de l'IA : si l'IA est « hard », le mouvement avec le plus grand nombre de points sera systématiquement choisi, mais si l'IA est « normal » ou « easy », le mouvement choisi sera alors relativement aléatoire (dans un certain intervalle).

2.6. ManageSaves.py

Le fichier ManageSaves.py contient le code nécessaire pour pouvoir créer un saveFile quand l'utilisateur décide de sauvegarder sa partie, et pour pouvoir charger la sauvegarde si elle existe déjà. Ce fichier a donc deux fonctions : createSaveFile et extractSaveFile.

La fonction createSaveFile prend en paramètres les gameState et gameSetup actuels. Si un fichier saveFile.txt existe déjà, il va alors être supprimé. Un (nouveau) fichier saveFile.txt sera alors créé. On y écrira ensuite ligne par ligne toutes les informations qui seront nécessaires pour relancer cette partie.

La fonction extractSaveFile ouvre le fichier saveFile.txt et lit ligne par ligne les informations qui sont notées. Cette fonction retourne alors les classes gameState et gameSetup complétées de toutes ses valeurs.

3. Détail des algorithmes « complexes »

3.1. L'initialisation de la grille

Détaillons ici l'algorithme de la fonction initializeGrid, présente dans le fichier GameState.py. Pour ce faire, il faut d'abord voir à quoi ressemble le plateau de jeu :

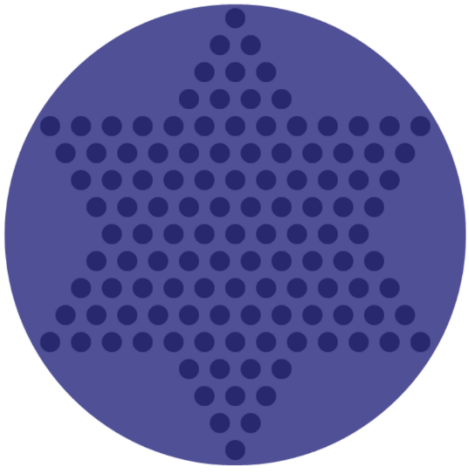

Figure 1 : La plateau de jeu vide

Le plateau de jeu est une forme d'étoile remplie de trous dans lesquels on peut placer des billes. Il a ensuite fallu que je trouve une façon de représenter chaque cercle par une variable, afin de plus tard pouvoir manipuler les valeurs de ces cercles.

La meilleure façon de représenter ces cercles a été pour moi de représenter le plateau de jeu par une grille. Mais une autre problème se pose : on voit que une ligne sur deux, les cercles ne sont pas sur les mêmes colonnes. J'ai donc décidé que chaque aurait soit seulement des cellules avec des colonnes paires, soit seulement des cellules avec des colonnes impaires, comme on peut le voir ci-dessous (les nombres en blanc représentent les colonnes, et les nombres en noir les lignes) :

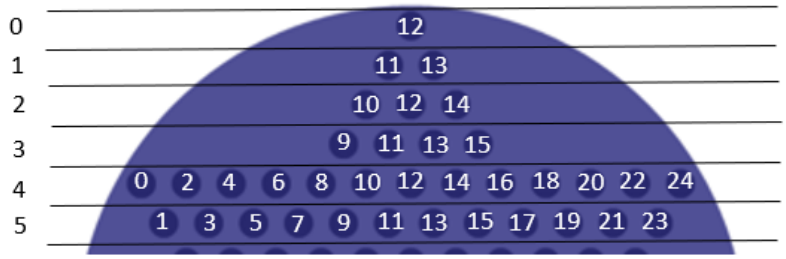

Figure 2 : Représentation du haut de la grille

Donc on voit que dans les lignes paires, il y a seulement des colonnes paires et dans les lignes impaires, il y a seulement des colonnes impaires.
On donnera ensuite une valeur à chaque cellule du tableau : les cases qui ne représentent pas un cercle (comme la case (1, 5) ou la case (2, 11) par exemple), auront une valeur de None, les « trous » auront une valeur de 0 et les billes du joueur 1 ou 2 auront une valeur de 1 ou 2 respectivement.

Codons maintenant tout cela. La fonction initializeGrid se fait en 3 étapes :

- Etape 0 : une grille de taille 25x17 a été créée dans l'initialisation de la classe (toutes les cellules ont la valeur None)
- Etape 1 : toutes les cellules qui sont à l'intérieur de l'étoile prennent la valeur 0.

On commence tout en haut de l'étoile au milieu (case (0, 12)) et on va avancer ligne par ligne. La variable nb_of_cells nous dit combien de cases font partie de l'étoile à ce moment-là. La variable mvt nous dit si la ligne d'après a plus ou moins de cases appartenant à l'étoile que la ligne actuelle.
Exemple :
On commence à la ligne 0.
nb_of_cells = 1 et mvt = 'more' (puisque la ligne 1 a plus de cellules dans l'étoile)
Comme mvt = 'more', on ajoute 2 à nb_of_cells et on passe à la ligne 1.
….
A la ligne 4 on passe directement à 25 cases. Donc mvt passe à 'less' et nb_of_cells = 25
…

- Etape 2 : On reparcourt la grille et on donne la valeur None aux cellules qui ont une ligne paire et une colonne impaire et aux cellules qui ont une ligne impaire et une colonne paire. On obtient ainsi la grille comme sur la figure 2.
- Etape 3 : On place les billes de chaque joueur à leur position de départ.

3.2. Trouver la position dans la grille à partir d'un click

On va ici détailler la fonction positionToCell présente dans le fichier GraphicMode.py dans la classe Game.

Quand un joueur clique, on récupère la position (x et y) en pixels grâce à la fonction GetLogicalPosition(wx.ClientDC(self)) incluse dans wxPython. On veut maintenant savoir si ces coordonnées correspondent à un cercle sur le plateau, et si oui, lequel.

J'ai créé pour cela la fonction positionToCell qui renvoie la cellule du cercle, si c'est bien un cercle qui a été cliqué, et None si le joueur a cliqué en dehors d'un cercle.

Etape 1 :

On trouve l'équivalence entre la position en pixels et la position dans la grille.
On trouve d'abord la ligne :
Chaque cellule a une hauteur de 35 pixels.
La première cellule est placée à 36 pixels du haut de la page.

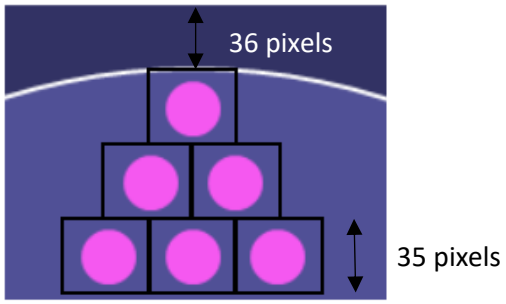

Figure 3 : Représentation des cases (haut de grille)

On trouve donc la ligne de la cellule en faisant le calcul : j = int((y-36)/35)

Ensuite, pour trouver la colonne, le calcul n'est pas le même selon si la ligne est paire ou impaire.
Chaque cellule a une longueur de 40 pixels.

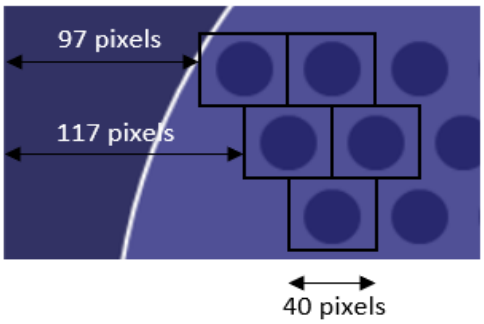

Figure 4 : Représentation des cases (côté de grille)

Donc si la ligne est paire, on fait le calcul i = int((x - 97)/40)*2
Si la ligne est impaire, on fait le calcul i = int((x - 117)/40)*2 + 1

On sait maintenant dans quelle case a cliqué le joueur, mais on ne sait pas encore si il a vraiment cliqué sur le cercle, ou alors l'espace qui l'entoure.

Etape 2 :

On calcule donc maintenant le centre x et y de la cellule dans laquelle le joueur a cliqué (qui le centre du cercle).
Ensuite, on calcule la distance entre la centre du cercle et l'endroit cliqué avec la formule :

$$dist = \sqrt{dist\_x^2 + dist\_y^2}$$

Si cette distance est inférieure à 13 (rayon du cercle) alors le joueur a cliqué dans le cercle.