Projet de Programmation Orienté Objet Python
Breakthrough

Présentation générale

- ### Le projet en trois phrases

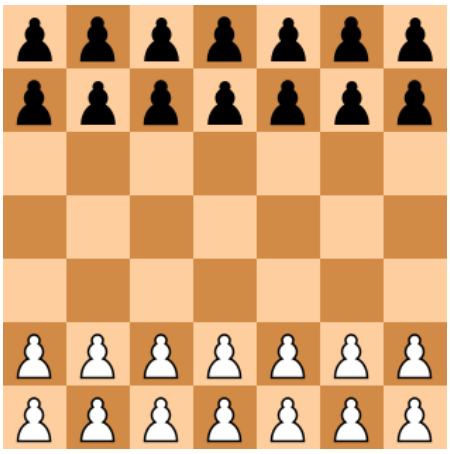
  L'objectif du projet est de réaliser une implémentation en Python 3 du jeu Breakthrough. C'est un jeu à deux joueurs sur un échiquier où chaque joueur contrôle deux rangées de pions, qui se déplacent toujours en avant (tout droit ou en diagonale) et qui prennent en diagonale. Le but est d'être le premier joueur à amener un pion à l'autre bout de l'échiquier.

- ### Les étapes de développement

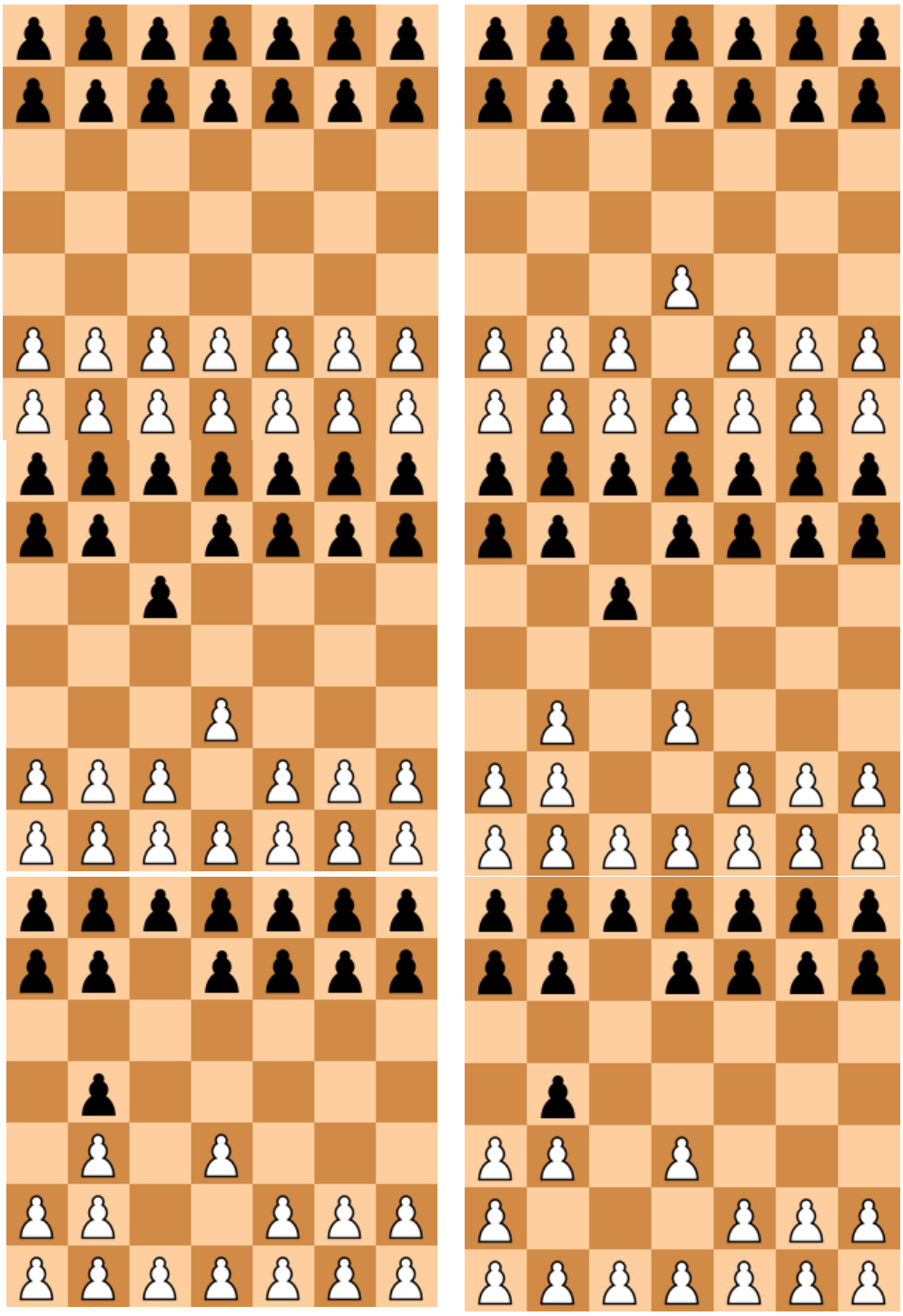
  La réalisation du projet est découpée en trois parties, chacune s'étalant sur environ un mois.
  Voici un résumé de ce qui sera développé dans chacune de celles-ci :
  — Partie 1 : Implémentation du jeu avec affichage en terminal.
  — Partie 2 : Différentes améliorations + ajout d'une IA basique.
  — Partie 3 : Réalisation d'une interface graphique pour le jeu à l'aide de la librairie PyQt.

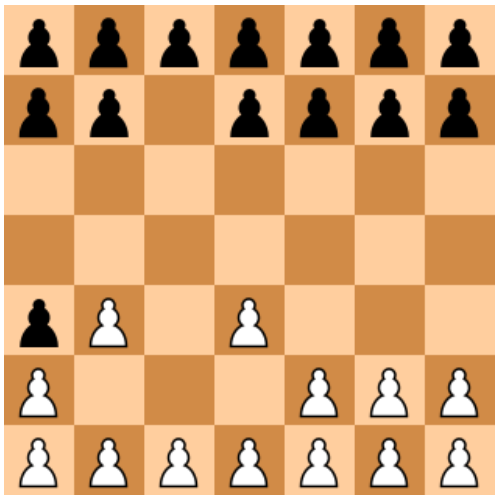
- ### Le jeu Breakthrough

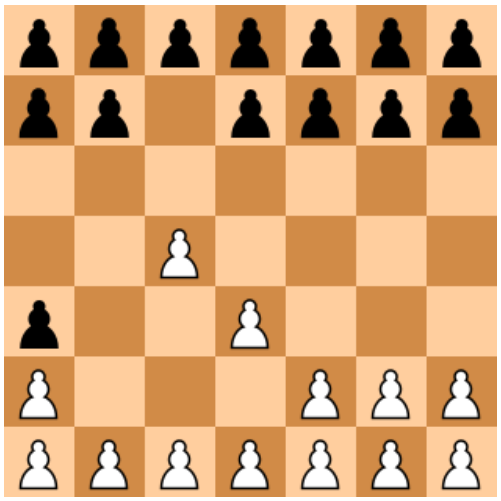
Le jeu Breakthrough a été inventé par Dan Troyka en 2000. Le jeu se joue habituellement sur un échiquier 7 × 7 ou 8 × 8 mais peut se jouer sur un échiquier de n'importe quelle taille. Pour cette introduction, nous allons utiliser un échiquier 7 × 7 pour les illustrations. Au départ, chaque joueur possède deux rangées complètes de pions :



Le joueur blanc commence, chaque joueur bouge un pion par tour. Un pion peut avancer d'une case, soit tout droit, soit en diagonale (gauche ou droite), si la case de destination est libre.
Un pion peut aussi prendre un pion adverse mais seulement via un mouvement en diagonale avant-gauche ou avant-droit (donc pas tout droit).
Le premier joueur qui réussit à amener un pion sur la dernière rangée du plateau pour lui (c-à-d la première rangée de l'adversaire) a gagné. Remarquons que, contrairement aux échecs, ici il n'y a pas de match nul possible.
Voici un exemple de début de partie :







Partie 1 : Jeu de base en terminal

La première partie du projet consistera à coder le jeu de Breakthrough sur le terminal, avec deux joueurs humains. Nous allons permettre de varier la taille du plateau.

1. Jeu principal

Votre fonction principale sera `main(n)`. Cette fonction lance une partie sur un plateau n × n et permet aux deux joueurs de jouer en alternance jusqu'à ce que l'un d'eux ait gagné. Le premier joueur joue avec les blancs (caractère W, pour 'white'), le second joue avec les noirs (caractère B, pour 'black'). À chaque fois qu'un joueur joue, il doit choisir une case où se trouve un de ses pions et la case où il souhaite déplacer celui-ci. Le plateau modifié est ensuite affiché à l'écran. Une fois qu'un joueur a gagné, un message de victoire est indiqué à l'écran après le dernier affichage du plateau.
Le jeu doit être lancé via la commande :

```
python3 partie1.py
```

Veillez à ce que votre fichier puisse être importé, mettez donc l'appel à la fonction main dans un test conditionnel `if __name__ == '__main__'`:

2. Affichage du plateau

Le plateau est affiché sous forme de matrice de taille n × n où les rangées sont numérotées de 1 à n de bas en haut et les colonnes seront déterminées par les lettres de l'alphabet commençant par a de gauche à droite, comme sur un plateau de jeu d'échecs.
Nous supposerons que 4 ≤ n ≤ 26, les 26 lettres de l'alphabet seront donc suffisantes (en pratique, les plateaux les plus couramment utilisés pour le jeu Breakthrough sont 7 × 7 et 8 × 8). Voici un exemple d'affichage à l'écran du plateau de départ avec un plateau de taille 7 × 7 :

```
  - - - - - - -
7 | B B B B B B B |
6 | B B B B B B B |
5 | . . . . . . . |
4 | . . . . . . . |
3 | . . . . . . . |
2 | W W W W W W W |
1 | W W W W W W W |
  - - - - - - -
  a b c d e f g
```

Les cases libres sont représentées par le caractère `.` et les pions blancs et noirs par respectivement les lettres W (pour 'white') et B (pour 'black'). En plus des barres verticales représentant les bords du plateau, les numérotations des lignes sont composées de 3 espaces qui sont remplis par les numéros. Cela veut dire que pour les numéros de 1 à 9, il y a un espace avant et après le numéro, tandis qu'à partir de la ligne 10, l'espace avant le numéro disparaît et est remplacé par le chiffre des dizaines de la ligne. Dans tous les cas, il y a un unique espace entre le numéro de la ligne et la barre verticale.
Les cases sont numérotées selon la convention des échecs, c'est-à-dire lettre de la colonne suivie du numéro de la ligne. Par exemple, la case a1 correspond à la case tout en bas à gauche du plateau.

3. Encodage du plateau

Le plateau est encodé sous forme d'une matrice board de taille n × n, c'est-à-dire une liste de listes composées chacune de n entiers, c'est-à-dire un par case : 0, 1 ou 2, selon que la case est libre ou occupée par un pion blanc ou noir, respectivement.
Nous utiliserons la convention suivante pour encoder les cases : Pour i entre 0 et n - 1, `board[i]` représente la ligne (rangée) numéro n-i, sous forme d'une liste, et pour j entre 0 et n-1, `board[i][j]` représente la case à la (n-i)ème ligne et (j+1)ème colonne (en comptant à partir de la gauche). Par exemple, pour n = 7 la case b2 correspond à l'entrée `board[5][1]`.

4. Format des mouvements

Un mouvement (ou "coup") est encodé de la manière suivante : `position du pion de départ>position d'arrivée du pion`, par exemple : `a7>a6`.

5. Fonctions à implémenter

Chaque fonction est présentée avec le nom, les paramètres de la fonction, dont chaque type est spécifié après les deux points, et le type de ce que renvoie la fonction après le `->`. Pour les connaisseurs, il s'agit d'une adaptation française la convention de typage de PEP python.
La notation `List[int]` correspond par exemple à une liste dont les éléments sont des variables de type `int`. La notation `Tuple[int,int]` correspond à un tuple contenant exactement deux éléments, chacun étant un entier.

Si `board` est la liste de listes représentant le plateau de jeu, pour une entrée `board[i][j]` nous appellerons la paire (i,j) *la paire d'indices correspondant* à cette entrée.

— `init_board(n : int) -> List[List[int]]` : Construit et renvoie la liste de listes qui représente le plateau de taille n × n de départ, selon l'encodage spécifié dans la Section 3 .

— `print_board(board : List[List[int]]) ->` None : Affiche le plateau de jeu, comme spécifié dans la section 2.

— `winner(board : List[List[int]]) -> int` ou None : Étant donné le plateau de jeu board, vérifie si l'un des deux joueurs a gagné. Renvoie 1 si le joueur blanc a gagné ; 2 si le joueur noir a gagné ; None si la partie n'est pas encore terminée.

— `is_in_board(n : int, pos : Tuple[int, int]) -> bool` : Renvoie `True` si la position pos est valide pour un plateau de taille n × n ; renvoie `False`, sinon. La position pos est une paire d'indices (i, j). Par exemple, sur un plateau 7×7, la position (0,5) est valide mais (1,7) ne l'est pas.

— `input_move() -> str` : Demande au joueur d'entrer un coup suivant le format décrit à la Section 4. Redemande un nouvel input tant que ce qui est entré par le joueur n'est pas interprétable comme un coup. Cette fonction ne doit pas vérifier que le coup est valide au sens des règles du jeu, ni que les cases décrites dans le coup sont bien dans le plateau. Ici on se contente de vérifier que la chaîne de caractères entrée par le joueur corresponde au format attendu, c-à-d : une lettre minuscule, suivi d'un ou deux chiffres, suivi du caractère >, suivi d'une lettre minuscule, suivi enfin d'un ou deux chiffres.

— `extract_pos(n : int, str_pos : str) -> pos : Tuple[int, int] ou None` : Traduit une position `str_pos` donnée en format notation échecs (ex : 'b5') en la paire d'indices (i, j) correspondante. Le paramètre n correspond à la taille du plateau.

— `check_move(board : List[List[int]], player : int, str_move : str) -> bool` : Prend en entrée le plateau de jeu et un mouvement proposé par l'un des joueurs selon le format décrit à la Section 4 (ex : 'a2>a3'), renvoie True si le coup est conforme aux règles du jeu et renvoie False sinon.

— `play_move(board : List[List[int]], move : Tuple[Tuple[int, int], Tuple[int, int]], player : int) -> None` : Modifie le plateau de jeu board en effectuant un coup donné pour un joueur donné. Cette fonction suppose que le coup respecte les règles du jeu (cette fonction sera donc utilisée une fois que la validité du coup entré par le joueur aura été vérifiée). Le coup est encodé sous forme d'une paire de positions ((i,j), (i', j')), (i,j) étant la position de départ et (i',j') la position d'arrivée, chaque position étant une paire d'indices.

— `main(n : int) -> None` : La fonction principale du programme. Elle contient d'abord une phase d'initialisation du plateau de jeu, de taille n × n, puis une boucle qui permet successivement aux joueurs de proposer des coups, jusqu'à ce que la condition de victoire soit satisfaite pour l'un des joueurs et que la partie s'arrête. Cette fonction doit aussi afficher l'état courant du plateau à chaque tour.

Partie 2 : Plateaux customisés et Intelligence Artificielle

Pour cette seconde partie, vous devrez implémenter des fonctionnalités supplémentaires, ainsi qu'une intelligence artificielle basique.

Génération d'un plateau de jeu sur base d'un fichier de configuration

La première amélioration à apporter consiste à permettre la création du plateau de jeu sur base d'un fichier texte. Ce fichier reprend :

— en ligne 1, deux chiffres indiquant le nombre de lignes et de colonnes, respectivement. Ces chiffres sont séparés par un espace ;

— en ligne 2 et 3, les positions des pions pour les joueurs blanc et noir, respectivement. Ces positions sont présentées au format `<lettre><chiffre>` et séparées par une virgule.

Voici un exemple de contenu d'un fichier de paramètre fonctionnel :

```
7 9
a1,b1,c1,d1,f1,g1,h1,i1,a2,b2,c2,d2,e2,f2,g2,h2,i2
a7,b7,c7,d7,e7,f7,g7,h7,i7,a6,b6,c6,d6,e6,f6,g6,h6,i6
```

Le chemin du fichier doit être donné en argument lors de l'appel du programme, comme suit :

```
python3 partie2.py board.txt
```

Si votre fichier se nomme board.txt. Si aucun fichier n'est donné en argument, un plateau par défaut, de taille 7×7, est créé. Les pions sont alors placés de manière à remplir les deux premières lignes de leur joueur respectif. Pour vérifier l'existence du fichier, vous pouvez utiliser la fonction : `os.path.isfile()` de la librairie os.

Sélection du pion

Pour rendre la navigation plus intuitive, nous vous demandons d'implémenter un dialogue de choix du pion à déplacer. A chaque tour d'un joueur, un message apparaîtra pour proposer au joueur de choisir un pion à jouer.

Attention : les directions mentionnées à partir d'ici font référence aux directions telles que vues par l'utilisateur du terminal, et pas en se mettant à la place des joueurs. La gauche fait donc mention à votre

gauche lorsque vous faites face à votre terminal, et aller vers le haut veut dire aller vers les lignes ayant un indice plus élevé selon la notation des échecs.

Le joueur pourra alors décider de sélectionner ce pion en entrant la touche y suivi de la touche entrée, ou pourra sélectionner un autre pion pour lequel des possibilités de déplacement valables existent.

Pour ce faire, les touches suivantes seront utilisées :

— l permettra de sélectionner le pion à droite ;
— j permettra de sélectionner le pion à gauche ;
— k permettra de sélectionner un pion sur une ligne inférieure ;
— i permettra de sélectionner un pion sur une ligne supérieure ;

Une fois toutes les possibilités de sélection de pion épuisées, l'énumération des pions de la ligne reprendra depuis l'extrémité opposée (par exemple, si le joueur arrive à l'extrémité gauche de la ligne et appuie sur j pour encore aller à gauche, le nouveau pion sélectionné sera celui de l'extrémité droite). Pour la sélection des pions des lignes supérieures et inférieures, le pion sélectionné sera celui qui est situé à la distance la plus faible du pion actuel. La distance utilisée pour cette opération est la distance Manhattan :

$$d = |x_1 - x_2| + |y_1 - y_2|$$

Si plusieurs pions sont situés à même distance, le pion le plus à gauche sera sélectionné. Notez que même si cette règle est arbitraire, l'évaluation se basant sur des tests automatiques, il est très important de la respecter.
Pour représenter le pion proposé dans le terminal, utilisez le caractère #. Une fois le pion sélectionné, celui-ci reprend son caractère initial (`W` ou B), et la position proposée prend à son tour la valeur #.
Attention, vous devez veiller à ne proposer que les pions qui peuvent bouger, et les mouvements possibles en l'état.

Intelligence artificielle basique

La dernière amélioration à apporter à votre jeu est la possibilité de jouer contre une intelligence artificielle basique. Cette IA sera une IA dite gloutonne : à chaque tour, elle avancera systématiquement le pion qui se situe le plus près de l'objectif (la ligne adverse). Si plusieurs pions sont situés à même distance, un pion sera choisi au hasard parmi ceux-ci.
Pour jouer contre l'IA, le programme devra être appelé avec l'argument `–ai` placé après le chemin du fichier encodant le plateau de départ (`board.txt` dans l'exemple ci-dessous) :

```
python3 partie2.py board.txt –ai
```

Fonctions à implémenter

— `init_board(file_path: str ou None) -> List[List[int]]` : Étant donné le chemin vers le fichier de configuration du plateau, adaptez cette fonction de sorte à ce qu'elle implémente le plateau décrit.

— `ai_select_peg(board: List[List[int]], player: int) -> tuple` : Permet à une IA de jouer automatiquement, de manière gloutonne, en avançant toujours le pion qui se trouve le plus proche de l'objectif. Si plusieurs pions sont à même distance et sont jouables, l'un d'eux est sélectionné aléatoirement. Pour sélectionner un pion aléatoirement, utilisez la fonction `choice()` de la librairie `random` sur la liste des pions éligibles **triée par ordre croissant de position**. La fonction renvoie un tuple donnant la position source sous forme d'une paire d'indices de la matrice board.

— `ai_move(board: List[List[int]], pos: Tuple(int, int), player: int) -> tuple` : Sélectionne aléatoirement une destination pour le pion choisi par l'IA. Pour sélectionner un mouvement aléatoirement, utilisez la fonction `choice()` de la librairie `random` sur la liste des mouvements possibles **triée par ordre croissant de position**. La fonction renvoie un tuple donnant la source et la destination du mouvement, sous forme de paires d'indices de la matrice board.

— `input_select_peg(board: List[List[int]], player: int) -> tuple` : Demande au joueur de sélectionner le pion à jouer. Les positions proposées par la fonction doivent obligatoirement être valides (un pion non jouable n'est jamais proposé au joueur). Le joueur sélectionne son pion grâce aux touche i (haut), k (bas), j (gauche), et l (droite), et valide son choix grâce à la touche y. La fonction renvoie un tuple donnant la position du pion, sous forme d'une paire d'indices de la matrice board.

— `input_move(board: List[List[int]], pos: Tuple(int, int), player: int) -> tuple` : Demande au joueur de sélectionner la destination du pion en position pos. La fonction renvoie un tuple donnant la source et la destination du mouvement, sous forme de paires d'indices de la matrice board.

— `find_closest_peg(current_peg: Tuple[int, int], next_line: List[Tuple[int, int]]) -> List[int]` : Étant donné un pion sélectionné et une rangée de pions, cette fonction renvoie la liste des indices des pions de la ligne ayant la distance la plus faible par rapport au pion choisi.

— `print_board(board : List[List[int]]) ->` None : Gardez le même format qu'en partie 1, mais faites attention aux nouvelles consignes qui auront un impact sur l'affichage.

— `main() ->` None : Comme précédemment, la fonction `main()` centralisera le déroulement du jeu. Ici, elle proposera au joueur 1 de choisir son coup, puis selon la configuration, soit prendra la place du joueur 2 via la fonction `ai_move()`, soit lui proposera de jouer également. Elle prendra également en charge la création d'un plateau de jeu par défaut, de taille 7 × 7 et avec les deux premières rangées de chaque joueur remplies de pions, si aucun fichier de configuration board.txt n'est donné.

Partie 3 : Interface graphique

La troisième partie du projet d'année consiste principalement à réaliser une interface graphique (Graphical User Interface) de votre programme. Pour implémenter l'interface graphique, il vous faut utiliser un package adapté. Bien que plusieurs tels packages existent pour Python3 (e.g. PyGTK, Tkinter ou encore PySide), vous utiliserez PyQt5. L'objectif est que vous appreniez à utiliser la documentation d'un package que vous ne connaissez pas, ce qui est une compétence nécessaire pour toute personne faisant du développement.

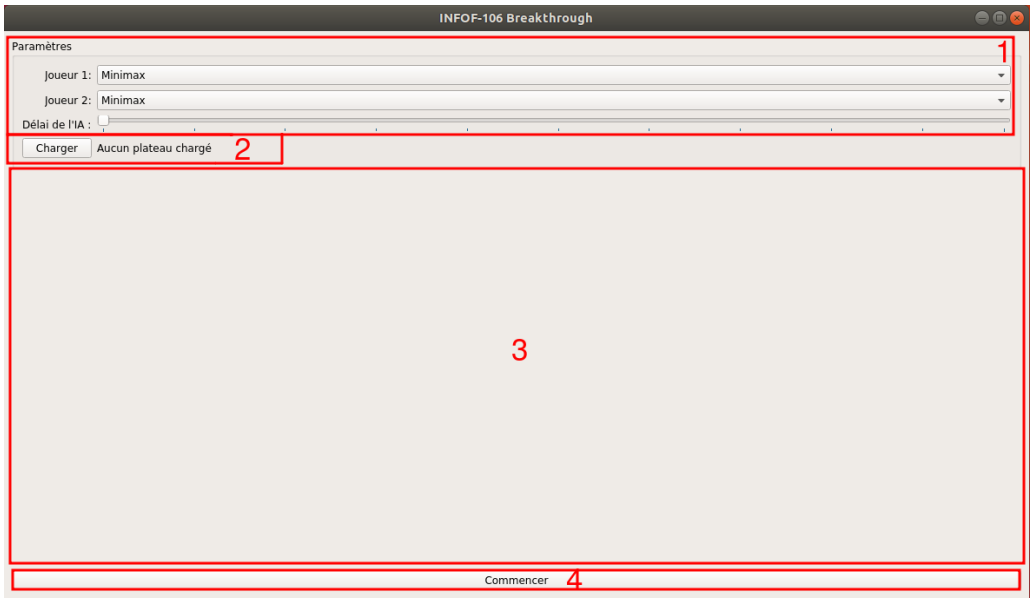
Structure et fonctionnement d'une GUI

Une GUI est typiquement composée d'un ensemble d'éléments nommés widgets agencés ensembles. Une fenêtre, un bouton, une liste déroulante, une boîte de dialogue ou encore une barre de menus sont des exemples de widgets typiques. Il peut également s'agir d'un conteneur (ou layout) qui a pour rôle de contenir et d'agencer d'autres widgets.
Une GUI va typiquement être structurée, de manière interne, sous la forme d'un arbre de widgets. Dans PyQt5, tout programme doit avoir un widget racine, qui est une instance de la classe QMainWindow, créant une fenêtre principale, dans laquelle on peut ensuite ajouter des widgets. Réaliser la structure visuelle d'une GUI (aussi appelée maquette) ne constitue que la moitié du travail, il faut que votre code des parties précédentes puisse être également lié à cette interface. Votre programme ne va donc plus être réduit à se lancer, effectuer des opérations et se terminer : à l'exécution, une GUI va réagir aux événements (par exemple, un clic d'un bouton). PyQt5 offre notamment la possibilité de lier le clic d'un bouton à l'appel d'une fonction donnée. Le principe d'une GUI est donc d'exécuter une boucle qui va traiter les événements jusqu'à la terminaison de l'application. Il vous faudra utiliser intelligemment ces possibilités pour mettre à jour l'affichage au besoin.
Remarque importante : PyQt propose un logiciel de création d'interfaces appelé Qt Designer. L'utilisation de Qt Designer est autorisée.

Exemple de maquette et éléments requis

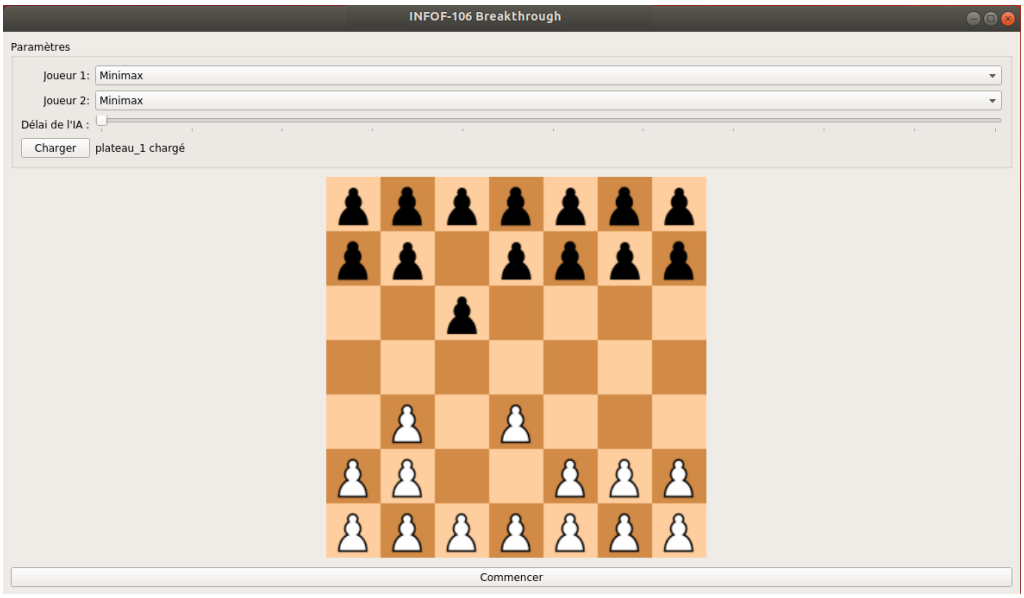
Vous trouverez une proposition de maquette pour votre programme sur la Figure 2. Cette dernière est découpée en 4 rectangles rouges contenant respectivement :

1. les options d'exécution (choix IA/Humain pour chaque joueur, et délai avant chaque coup d'une IA) ;
2. un widget permettant de charger un plateau de jeu ;
3. le canevas sur lequel le plateau de jeu est dessiné ;
4. le bouton pour démarrer une partie ou relancer une partie si une partie est déjà en cours.



Exemple de maquette.

La Figure montre un exemple d'exécution du programme. Vous y voyez le dessin sur le canevas et le bouton de (re)démarrage renommé.
Remarque : Notez bien que la maquette ci-dessus n'est qu'une proposition. Du moment que les éléments présentés ci-dessus figurent sur votre interface, vous êtes entièrement libres pour sa création et son esthétique, la créativité est de plus fortement encouragée. Votre maquette doit tout de même être claire et simple d'utilisation.



Exemple de maquette pendant le déroulement d'une partie.

Détails des éléments requis

Les éléments suivants doivent figurer d'une manière ou d'une autre sur votre interface (le choix du/des widget(s) pour y arriver vous est laissé libre) :

— le choix d'un joueur humain ou IA pour chaque joueur ;
— la sélection et le chargement d'un plateau de jeu ;
— le choix du délai avant chaque coup d'un joueur IA, ou dit autrement, le temps minimum avant de jouer un coup renvoyé par votre IA ;
— un dessin du plateau de jeu représentant les cases occupées par pions blancs et noirs ainsi que les cases vides ;
— la possibilité de lancer une partie ou relancer une nouvelle partie même si la partie en cours n'est pas finie ;

Lorsqu'un joueur humain désire jouer un coup, il doit cliquer sur l'un de ses pions (dont la sélection doit être visible graphiquement), ensuite cliquer sur une case vide valide (les différentes possibilités de déplacement doivent également être visibles graphiquement) sur laquelle le pion sélectionné se déplacera.
Le temps pris entre deux coups de l'IA peut être modifié pendant une partie mais les joueurs eux ne peuvent plus l'être après le démarrage de la partie. Le seul moyen de changer le mode de jeu d'un des joueurs est d'arrêter la partie en cours, de modifier le mode de jeu et de relancer une partie.
Lorsqu'une partie se finit, le joueur gagnant doit être signalé, et plus aucune action sur le plateau de jeu ne doit être acceptée.

Utilisation de classes

Le package PyQt5 est entièrement conçu selon la programmation orienté objet (POO), il vous est dès lors demandé d'implémenter des classes de manière à pouvoir interfacer votre code des parties précédentes avec votre interface graphique. Veillez bien à respecter toutes les règles de bonne pratique associées que vous avez vues au cours de programmation. De plus, pensez à structurer votre code proprement. En particulier, séparez votre code en fichiers. Vous devrez alors importer le code dans votre fichier partie3.py à l'aide du mot-clef import.