# Info-F-106 : Projet d'Informatique
## Jeu des amazones

Gwenaël Joret Charlotte Nachtegael Robin Petit Cédric Simar

version du 26 avril 2021

## Présentation générale

### Le projet en trois phrases

L'objectif du projet est de réaliser une implémentation en Python 3 du jeu des amazones. C'est un jeu à deux joueurs sur un grand échiquier $10 \times 10$ où chaque joueur contrôle quatre *amazones* qui se déplacent comme les reines aux échecs, et qui peuvent tirer des flèches pour bloquer des cases. Le but est de bloquer complètement le joueur adverse.

### Les étapes de développement

La réalisation du projet est découpée en quatre parties, chacune s'étalant sur environ un mois. Voici un résumé de ce qui sera développé dans chacune de celles-ci :

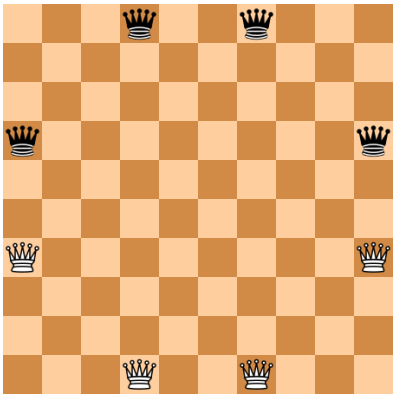
— Partie 1 : Jeu avec affichage en terminal.
— Partie 2 : Ajout d'une IA basique.
— Partie 3 : Différentes améliorations du jeu et de l'IA.
— Partie 4 : Réalisation d'une interface graphique pour le jeu à l'aide de la librairie `PyQt`.

### Le jeu des amazones

Le jeu des amazones a été inventé par Walter Zamkauskas en Argentine en 1988. C'est un jeu à deux joueurs (blanc et noir) qui se joue sur un grand échiquier $10 \times 10$. Notons que ce n'est pas l'échiquier standard des échecs, qui lui est de dimension $8 \times 8$.

Chacun des deux joueurs possède quatre amazones. Ce sont des pièces qui se déplacent comme les reines aux échecs : horizontalement, verticalement, ou diagonalement, sans limite sur le nombre de cases. La position de départ est la suivante [1] :

1. Les dessins des différentes pièces proviennent de Wikipedia (dessins pour les échecs).



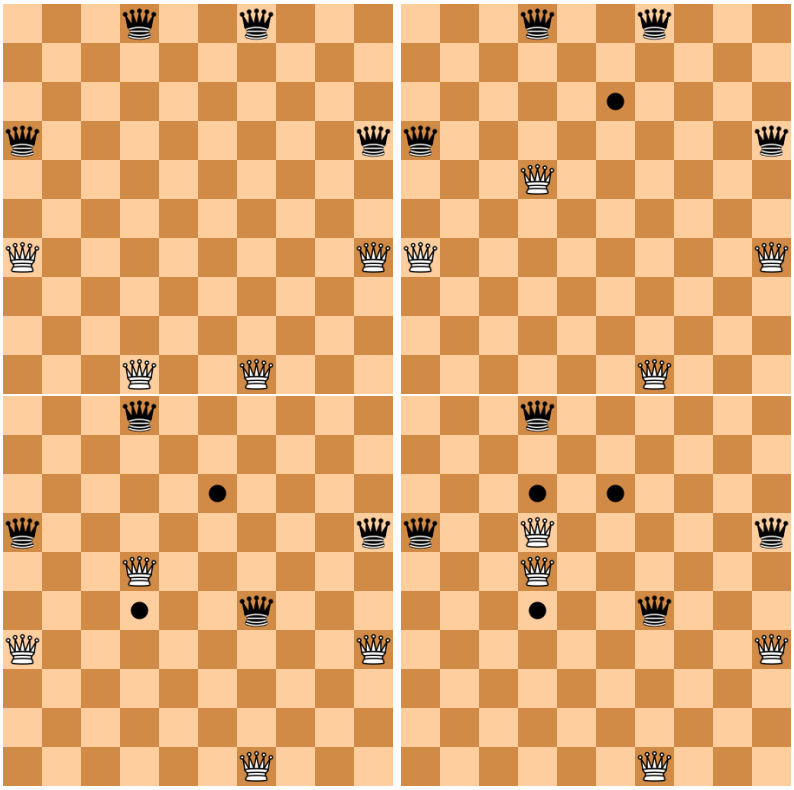
Lorsqu'une amazone se déplace, elle tire ensuite une flèche à partir de sa position d'arrivée. La flèche se déplace elle aussi comme les reines aux échecs. Contrairement à ce que l'on pourrait penser, la flèche n'est pas envoyée vers une pièce adverse. Au contraire, la flèche doit terminer sa trajectoire sur une case vide, et ne peut pas passer au-dessus de pièces ou d'autres flèches, il faut que toutes les cases de sa trajectoire soient vides. Ces flèches formeront des obstacles pour les amazones : une amazone ne peut pas passer au dessus, ni s'arrêter sur une case contenant une flèche.

Les deux joueurs jouent tour à tour ; c'est le joueur blanc qui commence. Lors de son tour, un joueur doit effectuer les deux actions suivantes, dans l'ordre :

1. choisir une de ses amazones et la déplacer d'au moins une case, et
2. lui faire tirer une flèche.

Si au tour d'un joueur, celui-ci est bloqué, c-à-d qu'il ne peut déplacer aucune de ses amazones, alors ce joueur a perdu. En d'autres mots, c'est le dernier joueur qui a pu déplacer une amazone qui gagne la partie. Il n'y a pas de match nul.

Voici un exemple de début de partie :



Une remarque concernant la fin de partie : Il est possible que le dernier joueur qui ait pu déplacer une de ses amazones bloque le joueur adverse *et* lui-même lorsqu'il tire sa dernière flèche. Cela ne change pas le fait qu'il gagne la partie.

## Organisation

Pour toute question portant sur ce projet, n'hésitez pas à rencontrer le titulaire du cours ou la personne de contact de la partie concernée.

**Titulaire.** Gwenaël Joret – gjoret@ulb.ac.be – O8.111

**Assistants.**
*Parties 1 et 2 :* Charlotte Nachtegael – charlotte.nachtegael@ulb.be – N8.213
*Partie 3 :* Cédric Simar – cedric.simar@ulb.ac.be – N8.212
*Partie 4 :* Robin Petit – robpetit@ulb.ac.be – O8.210

## Consignes générales

— L'ensemble du projet est à réaliser en `Python 3`.
— Le projet est organisé autour de quatre grandes parties
— En plus de ces quatre parties à remettre, chaque étudiant devra remettre un rapport final.
— La répartition des points est la suivante :
  — Partie 1 – correction automatique : 20 points

  - Partie 2 – 20 points (2/3 des points : correction automatique ; 1/3 des points : utilisation de git)
  - Partie 3 : 20 points
  - Partie 4 : 20 points
  - Rapport : 20 points
  - **Total : 100 points**
- Les quatre premières parties devront être remises sur GitHub Classroom.
- Après chacunes des trois premières parties, un correctif sera proposé. Vous serez libre de continuer la partie suivante sur base de ce correctif mais nous vous conseillons de plutôt continuer avec votre travail en tenant compte des remarques qui auront été faites.
- Les « Consignes de réalisation des projets » (cf. `http://www.ulb.ac.be/di/consignes_projets_INFO1.pdf`) sont d'application pour ce projet individuel. *(Exception : Ne tenez pas compte des consignes de soumission des fichiers, des consignes précises pour la soumission via GitHub Classroom seront données).* Vous lirez ces consignes en considérant chaque partie de ce projet d'année comme un projet à part entière. Relisez-les régulièrement !
- Si vous avez des questions relatives au projet (incompréhension sur un point de l'énoncé, organisation, *etc.*), n'hésitez pas à 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 !**

Veuillez noter également que le projet vaudra **zéro** sans exception si :

- le projet ne peut être exécuté correctement via les commandes décrites dans l'énoncé ;
- les noms de fonctions (et de vos scripts) sont différents de ceux décrits dans cet énoncé, ou ont des paramètres différents ;
- vous avez modifié les fichiers de tests avec l'intention de tricher ;
- à l'aide d'outils automatiques spécialisés, nous avons détecté un plagiat manifeste (entre les projets de plusieurs étudiants, ou avec des éléments trouvés sur Internet). Nous insistons sur ce dernier point car l'expérience montre que chaque année une poignée d'étudiants pensent qu'un petit copier-coller d'une fonction, suivi d'une réorganisation du code et de quelques renommages de variables passera inaperçu... Ceci sera sanctionné d'une note nulle pour l'entièreté du projet pour toutes les personnes impliquées, ainsi que d'éventuelles autres sanctions. Afin d'éviter cette situation, veillez en particulier à ne pas partager de bouts de codes sur forums, Facebook, etc.

## Soumission de fichiers

La soumission des fichiers se fait via un repository individuel par partie du projet sur la plateforme GitHub Classroom. Le lien pour s'inscrire au projet sur GitHub Classroom ainsi que des explications concernant l'utilisation de l'outil Git sont disponibles sur **les slides de présentation de Git / GitHub Classroom** sur l'UV.

Une remarque concernant les retards : Contrairement à d'autres cours d'informatique, il n'est ici pas possible de remettre une de vos parties en retard. Le système de versioning offert par Git vous permet de constamment mettre à jour la version de votre code sur le serveur de GitHub Classroom. Vous êtes d'ailleurs **fortement encouragé** à le faire régulièrement lorsque vous travaillez sur une partie. Ceci vous permet de garder une copie de chaque version intermédiaire de votre travail, et nous permet à nous, en tant qu'enseignants, d'avoir une idée de la régularité de votre travail. Pour l'evaluation d'une partie, vous ne devez pas indiquer quelle est la version

"finale" de votre code, nous prendrons simplement la dernière version uploadée sur le repository avant la date et heure limite (toute version uploadée après sera ignorée).

### Objectifs pédagogiques

Ce projet *transdisciplinaire* permettra de solliciter diverses compétences.

— *Des connaissances vues aux cours de programmation, langages, algorithmique ou mathématiques.* 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.* Les étudiants seront invités à les étudier par eux-mêmes, aiguillés par les *tuyaux* fournis par l'équipe encadrant le cours. Il s'agit entre autres d'une connaissance de base des interfaces graphiques en `Python 3`.

— *Des compétences de communication.* Après la partie 4, les étudiants remettront un rapport expliquant leur analyse, les difficultés rencontrées et les solutions proposées. Le rapport devra être rédigé en LaTeX. Ce sera l'occasion pour les étudiants de se familiariser avec ce langage, utilisé pour la rédaction de documents scientifiques. Une orthographe correcte sera exigée.

En résumé, les étudiants devront démontrer 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.

# Bon travail !

# 1 Partie 1 : Jeu de base en terminal

La première partie du projet consistera à coder le jeu des Amazones sur le terminal, avec deux joueurs humains. Par rapport au jeu standard, nous allons également permettre de varier la taille du plateau, ainsi que le nombre et les positions de départ des reines (amazones).

## 1.1 Remarque préliminaire concernant la correction

Une spécificité de cette première partie du projet est que la correction sera faite automatiquement via une série de tests sur GitHub Classroom, c'est ce qu'on appelle de *l'autograding*. L'entièreté de ces tests est mise à votre disposition, vous pourrez voir à tout moment les tests réussis et ratés par votre code, et la note (sur 20) associée à votre code. Ainsi vous aurez un feedback direct sur votre partie 1.

L'autograding est une nouveauté introduite cette année. Nous espérons que cela vous aidera à bien démarrer ce projet d'année, qui est assez conséquent. Le fonctionnement de l'autograding a été décrit en détail lors de la séance de présentation du projet du 19 octobre 2020 sur Teams. Cette séance a été enregistrée, n'hésitez pas à la revisionner !

## 1.2 Constantes mises à disposition

Vous trouverez dans votre repository de départ un fichier contenant toutes les constantes nécessaires pour le jeu : les caractères pour les différents joueurs, flèches et cases libres, ainsi que les messages d'erreur.

Pour les utiliser, vous devez ajouter en haut de votre fichier `partie1.py` :

```
from source.constantes import *
```

Remarque : le `source.` est requis pour permettre de lancer le programme à partir de la racine du repository (voir plus bas), ce qui est nécessaire pour l'autograding sur GitHub Classroom.

Certains des messages contiennent des {}, où vous pouvez ajouter du texte à l'aide de la méthode `format()`. Exemple : si la variable `x = "Joueur {} est vaincu !"`, et la variable `joueur = 1`, alors `print(x.format(joueur))` imprimera `Joueur 1 est vaincu !`

## 1.3 Jeu principal

Votre fonction principale sera `main(n, positions_noir, positions_blanc)` où `n` correspond à la taille du plateau (c-à-d, un plateau $n \times n$), `positions_noir` est une liste de strings des positions des reines noires et `positions_blanc` est une liste de strings des positions des reines blanches.

Cette fonction lance une partie 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 ○), le second joue avec les noirs (avec le caractère ●). À chaque fois qu'un joueur joue, il doit choisir une case où se trouve une de ses reines, la case où il souhaite la déplacer, et enfin la case où il veut mettre sa flèche après s'être déplacé. 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. La fonction doit renvoyer 1 si le joueur blanc a gagné ou 2 si c'est le joueur noir.

Le jeu doit être lancé via la commande

```
python3 -m source.partie1
```

depuis le *dossier racine* de votre repository GitHub, donc celui où vous voyez les dossiers source et tests.[2] Veillez à ce que votre fichier puisse être importé, mettez donc l'appel à la fonction `main` dans un test conditionnel `if __name__ == '__main__':`

## 1.4 Affichage du plateau

Le plateau est affiché sous forme de matrice de taille $n \times n$ où les rangées sont numérotées de 1 à `n` de bas en haut et les colonnes seront déterminées par les lettres commençant par a de gauche à droite, comme sur un plateau de jeu d'échecs. Voici un exemple d'affichage à l'écran d'une partie en cours avec un plateau de taille 10 :

```
10 . . . ● . . ● . . .
 9 . . . . . . . . . .
 8 . . . . . . . . . .
 7 ● . . . . . . . . ●
 6 . . . . . . . . . .
 5 . . . . . . . . . .
 4 ○ . . . . . . . . ○
 3 . . . . . . . . . .
 2 . . . . . . . . . .
 1 . . . ○ . . ○ . . .
   a b c d e f g h i j
```

Les cases libres sont représentées par un `.` et les flèches par un `X`. Les numérotations des lignes sont composées de 3 espaces qui sont remplis par les numéros. Cela veut dire que les numéros de 1 à 9 sont séparés par deux espaces du début du plateau et à partir de la ligne 10, d'un seul espace. Les cases composant le plateau sont séparées par un espace.

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.

## 1.5 Encodage du plateau

Le plateau est encodé sous forme d'une matrice de taille $n \times n$, c'est-à-dire une liste de listes, composée de strings, correspondant soit à `.`, `X`, ○, ● selon que la case est libre, occupée par une flèche, une reine blanche ou une reine noire respectivement.

Nous utiliserons la convention suivante pour encoder les cases : Pour `i` entre 0 et $n-1$, `plateau[i]` représente la ligne (rangée) numéro $i+1$, sous forme d'une liste, et pour `j` entre 0 et $n$, `plateau[i][j]` représente la case à la $(i+1)$-ème ligne et $(j+1)$-ème colonne (en comptant à partir de la gauche). Par-exemple, la case `b8` correspond à l'entrée `plateau[7][1]` de la matrice de taille $10 \times 10$.

Notez bien que cela veut dire qu'on affiche la dernière liste de la matrice en premier !

## 1.6 Format des coups

Le coup est sous forme de string suivant la règle suivante : `reine de départ>position reine arrivée>position flèche`, par exemple : `a7>b7>a8`

2. Cette façon inhabituelle de lancer l'interpreteur python est nécessaire ici, pour permettre l'autograding sur GitHub Classroom.

Chaque coup doit être encodé sous forme de lettre minuscule + nombre représentant des positions possibles sur le plateau et séparés par le caractère `>`. Si le format du coup n'est pas valide, vous imprimez le message `ERREUR_COUP`.

S'il n'y a pas de reine à la case de départ, vous imprimez le message `ERREUR_REINE`.

Si la destination de la reine ou si l'endroit où on veut tirer la flèche n'est pas en accord avec les mouvements dans le jeu d'échecs (direction droite suivant les directions cardinales N, S, E, O, NE, NO, SE, SO et aucun obstacle, reine ou flèche, sur le chemin), alors vous imprimez le message `ERREUR_CHEMIN`. Faites bien attention à vérifier que le chemin de la flèche est valide depuis la position où arrive la reine.

Vous devez continuer de demander un coup au joueur jusqu'à ce qu'il entre un coup valide, à l'aide du message `MESSAGE_COUP`.

## 1.7 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.

— `main(n : int, positions_noir : liste de str, positions_blanc : liste de str) -> int` : Fonction principale du jeu. Cette fonction lance une partie et permet aux deux joueurs de jouer en alternance jusqu'à ce que l'un d'eux ait gagné, en commençant avec les blancs. Renvoie un int, correspondant à 1 si le joueur blanc gagne et à 2 si c'est le joueur noir.

— `construire_plateau(n : int, positions_noir : liste de str, positions_blanc : liste de str) -> liste de listes de str` : Construit la liste de listes qui constitue le plateau de taille $n \times n$ avec les caractères `JOUEUR_BLANC` aux positions de la liste `positions_blanc` et les caractères `JOUEUR_NOIR` aux positions de la liste `positions_noirs`, comme spécifié dans la Section 1.5.

— `string_plateau(plateau : liste de listes de str) -> str` : Crée le string pour imprimer le plateau, comme spécifié dans la Section 1.4.

— `coup_to_tuple(coup : str, taille : int) -> (int, int) ou None` : Traduit le coup en format string en tuple (ligne, colonne). La fonction renvoie `None` si le coup n'est pas valide. Le paramètre `taille` correspond à la taille du plateau et le paramètre `coup` représente une case du plateau sous forme de string, tel que `a8` ou `e10`.

— `mouvements_possibles(source : str, plateau : liste de listes de str) -> liste de str` : Donne les mouvements possibles depuis la case source, les mouvements en question étant toutes les cases en ligne droite (directions cardinales N, S, E, O, NE, NO, SE, SO) jusqu'à la rencontre d'un obstacle (reine ou flèche), sous la forme de string. Le paramètre `source` représente une case du plateau sous forme de string. La liste renvoyée doit être triée dans l'ordre croissant (alphabétique).

— `coup_joueur(joueur : int, positions_joueur : liste de str, plateau : liste de listes de str) -> None` : Fonction pour jouer un coup par un joueur

humain, vérifie l'input et applique le coup si valide. C'est dans cette fonction que vous devez imprimer les messages d'erreur et demander le coup en input au joueur humain. Le paramètre `joueur` est 1 ou 2 selon si le joueur blanc ou noir respectivement est en train de jouer, `positions_joueur` est la liste des positions où se trouvent les reines du joueur en train de jouer, sous le format string. Si le coup est valide, vous devez modifier les listes `plateau` et `positions_joueur`, mais ne rien renvoyer ! Il est important de noter que pour modifier la liste des positions, vous devez retirer l'ancienne position de la reine et ajouter **à la fin de la liste** la nouvelle position. Indice : pour vérifier si vos coups sont valides, utilisez les fonctions `coup_to_tuple` et `mouvements_possibles`.

— `fin(plateau : liste de listes de str, positions_noir : liste de str, positions_blanc : liste de str, joueur : int) -> (int, int, int)` : Fonction qui détermine si le jeu est terminé et qui renvoie un tuple qui correspond au `(gagnant, score du joueur blanc, score du joueur noir)`. Le `gagnant` est 1 si joueur blanc gagne, 2 si joueur noir gagne, 0 si pas de gagnant. Les `scores` sont à 0 par défaut. Si un joueur gagne, son score est mis à 1. Les paramètres `positions_noir` et `positions_blanc` correspondent aux positions des reines des deux joueurs et le paramètre `joueur` correspond au joueur qui vient de joueur (1 pour blanc, 2 pour noir). Le jeu s'arrête lorsqu'un des deux joueurs ne sait plus se déplacer. Le dernier joueur à avoir pu se déplacer gagne (même si les deux joueurs sont bloqués). Les scores des joueurs blancs et noirs sont présents dans les résultats pour être utilisés pour une intelligence artificielle lors de la partie 2.

Vous êtes naturellement invités à implémenter des fonctions supplémentaires si vous le désirez, mais vous devez obligatoirement implémenter telles que demandées les fonctions ci-dessus. Ces fonctions sont celles qui seront testées sur GitHub Classroom et ne laisseront aucune place à l'erreur. Faites également attention à ne pas utiliser de variables globales, comme celles-ci pourraient vous faire échouer les tests.

## 1.8 Librairies permises

— `sys`
— `os`

Aucune autre librairie ne sera autorisée.

## 1.9 Consignes de remise

Tout d'abord, insistons sur l'importance de respecter toutes les consignes ci-dessus et ci-dessous : **tout manquement aux consignes sera sanctionné d'une note nulle.**

Afin de créer votre repository pour votre partie 1 sur GitHub Classroom, vous devrez utiliser le lien suivant : `https://classroom.github.com/a/ZZevLSVo`

Voici ensuite les étapes à suivre :

1. Sélectionnez votre Prénom Nom dans la liste. Si vous ne le trouvez pas, vous pouvez ignorer cette partie et contacter l'assistant chargé de la partie. Attention, la liste est ordonnée selon le prénom (et non le nom) !
2. Attendez quelques minutes avant de rafraîchir la page pour obtenir le lien vers votre projet. Une fois obtenu, assurez-vous d'avoir bien les dossiers : .gihtub, source et tests.
3. **Forkez** le repository sur votre compte personnel (voir Fig 1). Cela signifie faire une copie du projet que vous pouvez modifier autant que vous le voulez.

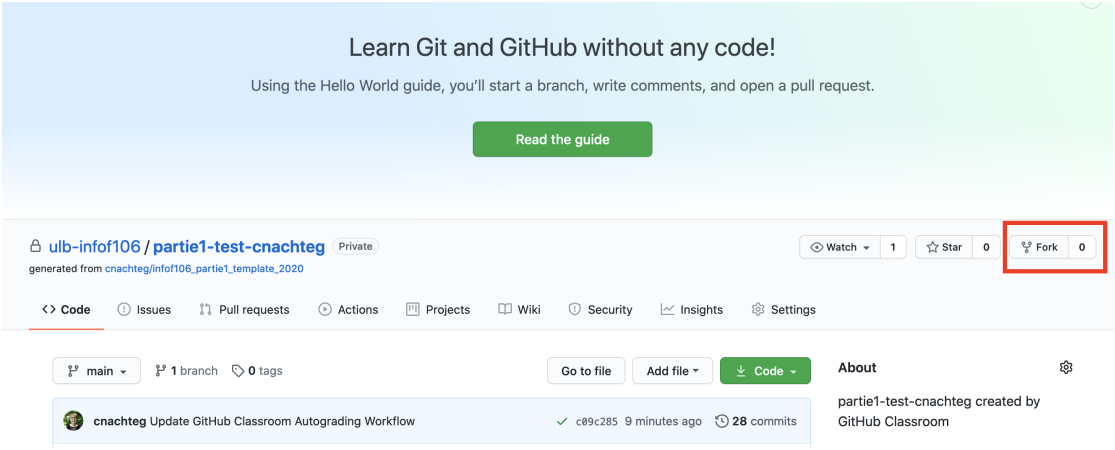



FIGURE 1 – Où trouver le bouton fork sur votre repository GitHub

4. Sur **votre repository personnel**, que vous pouvez reconnaître car votre github id précède le nom de votre repository et `ulb-infof106` comme sur la figure, cliquez sur l'onglet `Actions` et autorisez les `Workflows`. Cela vous permettra d'avoir les tests automatiques sur votre repository, afin de voir les points que vous obtiendrez, que vous trouverez après chaque push dans l'onglet `Actions`.
5. À présent, vous pouvez cloner ce repository et faire tous les commits et push dessus.
6. **AVANT LA DEADLINE**, n'oubliez pas de faire **UN** pull request (onglet sur le repository sur le web) pour transférer votre projet sur l'assignment. Nous ne regarderons que ce qui se trouve sur votre repository dans l'organisation `ulb-infof106`.

L'entièreté de votre code doit se trouver dans un fichier `partie1.py` (avec une première lettre minuscule). **Votre nom, numéro de matricule et section doivent être indiqués en commentaire au début du fichier `partie1.py`.**

Le jeu doit commencer en lançant la commande `python3 -m source.partie1` depuis le dossier GitHub, donc celui où vous voyez les dossiers source et tests. Veillez à ce que votre fichier puisse être importé, mettez donc l'appel à la fonction `main` dans un test conditionnel `if __name__ == '__main__':`.

À travers l'entièreté de votre projet, veillez à respecter à la lettre la signature des fonctions (nom + paramètres + type de la valeur retournée) ainsi que la casse des caractères (respect des majuscules et minuscules).

**Uploadez régulièrement votre code** sur le serveur ! Une bonne habitude est de le faire systématiquement à la fin de chaque session de travail, par exemple :

```
git commit -am 'courte description des modifications'
git push
```

La bonne utilisation de git fait partie de la note de ce projet.

Astuce : pour ne pas devoir entrer votre login et mot de passe github à chaque push, vous pouvez utiliser la commande suivante :

```
git config credential.helper store
```

Echéance pour la remise du fichier `partie1.py` sur GitHub Classroom dans le dossier source : **dimanche 22 novembre à 22h**.

**Remarques importantes :**

— Il n'y a aucun retard possible, GitHub Classroom n'acceptera aucun `commit` passé cette heure. La version de votre code qui sera évaluée sera la dernière version sur GitHub Classroom.

— Si vous rencontrez des problèmes avec l'utilisation de git, nous vous invitons à relire les slides d'introduction à git disponibles sur l'UV, et à poser vos éventuelles questions qui subsisteraient aux guidances ou aux assistants. Dans tous les cas, posez vos questions **bien à l'avance**, pas deux-trois jours avant l'échéance. Pour rappel, l'usage de git sera évalué, et en particulier la régularité des `commit` et `push`.

— Chaque année, quelques étudiants attendent le dernier moment pour apprendre à utiliser git, et rencontrent des problèmes avec git le weekend de la remise. Il est inutile d'envoyer le projet par email et d'expliquer les problèmes rencontrés, seuls les projets remis via GitHub Classroom sur l'organisation `ulb-infof106` seront évalués.

— Faites bien attention à utiliser le lien GitHub Classroom ci-dessus pour créer votre repository et à forker celui-ci, et à ne pas créer vous-même un repository GitHub standard de votre côté. Ces derniers sont publics (!) par défaut, vous partageriez donc votre code avec tous les internautes, ce qui serait sanctionné d'une note nulle. Nous insistons sur ce dernier aspect car quelques cas se sont présentés l'année passée lors de la partie 1.

— Ne touchez pas au dossier `tests` du repository ! Nous allons vérifier l'intégrité des fichiers tests après la remise et s'il s'avère que vous les avez modifiés pour réussir les tests, vous aurez une note nulle.

— Nous allons effectuer des tests de plagiat sur vos codes. Si nous détectons du plagiat entre plusieurs personnes, toutes les personnes concernées recevront **une note nulle pour l'entièreté du projet d'année**, ils ne pourront donc pas continuer le projet. Rappelons qu'outre une note nulle pour le cours concerné, un plagiat à l'université peut donner lieu à des sanctions supplémentaires, allant jusqu'à l'exclusion du programme d'étude. Soyez donc extrêmement vigilants lorsque vous discutez du projet d'année avec vos collègues et **ne partagez jamais un bout de code**, même de 2-3 lignes.

**Bon travail !**

**Personne de contact : Charlotte Nachtegael – charlotte.nachtegael@ulb.be – N8.213**

# 2 Partie 2 : Plateaux customisés et IA

La seconde partie du projet consiste à ajouter la lecture de plateaux personnalisés, ainsi que d'implémenter une IA très basique comme adversaire en tant que joueur noir.

## 2.1 Plateau customisé

Le plateau est maintenant créé à partir des paramètres trouvés dans un fichier dont le chemin est donné en ligne de commande. Le fichier est composé de quatres lignes avec :

1. Taille du plateau, int
2. Positions des reines noires, en format case en string, séparées par des virgules
3. Positions des reines blanches, en format case en string, séparées par des virgules
4. Positions des flèches déjà tirées, en format case en string, séparées par des virgules

Par exemple :

```
10
a7,j7,d10,g10
d1,g1,a4,j4
a3,b3,b4,j8,i8,i7
```

Vous devrez maintenant récupérer le fichier sur la ligne de commande à l'aide de la fonction `argv` de la librairie `sys` (`sys.argv`), lire le contenu du fichier et construire un plateau avec comme ajout par rapport à la partie 1 les flèches qui sont déjà présentes sur le plateau.

La nouvelle ligne de commande pour lancer votre script sera :

```
python3 -m source.partie2 nom_de_fichier
```

## 2.2 IA Minimax

Pour cette partie, nous allons implémenter une intelligence artificielle qui va suivre le principe de l'algorithme Minimax, un algorithme récursif.

Selon l'état d'un plateau, on trouve le meilleur état en simulant tous les scénarios possibles et en choisissant le meilleur scénario. Le meilleur scénario est celui avec la meilleure situation finale en assumant que :

— on fait toujours le meilleur mouvement pour **maximiser** son score et
— l'adversaire fait toujours le mouvement qui est le meilleur pour lui (et donc le pire pour nous), **minimisant** notre score.

Ce principe de **maximisation** et de **minimisation** est la raison pour laquelle l'algorithme se nomme Minimax.

Le score est toujours calculé selon le point de vue de l'IA, même lorsque c'est le tour de simuler l'autre joueur :

— 1 pour une victoire
— -1 pour une défaite
— 0 s'il n'y a pas de gagnant

L'IA dans notre cas étant le joueur noir, cela signifie qu'une défaite est équivalente à `-score_blanc` obtenu grâce à la fonction `fin` et qu'une victoire est égale à `score_noir`.

L'algorithme suit le pseudocode suivant :

```
function MINIMAX(plateau, positions_joueurs, profondeur, maxi)
    if profondeur == 0 ou le jeu est fini then
        return (None, score du plateau)
    end if
    if maxi then
        meilleurscore = -1000
        meilleurcoup = []
        for reine dans positions du joueur noir do
            for mouvement de reine dans mouvements possibles do
                Bouge la reine sur le plateau
                Modifie la position de la reine dans les positions du joueur noir
                for fleche dans mouvements de fleche possibles do
                    Met la flèche sur le plateau
                    score = minimax(plateau, positions_joueur, profondeur - 1, FALSE)[1]
                    coup = position reine départ > mouvement reine > fleche
                    if meilleurscore < score then
                        meilleurscore = score
                        meilleurcoup = [(coup,score)]
                    else if meilleurscore == score then
                        Ajoute le coup dans les meilleurs coups
                    end if
                    Efface la flèche sur le plateau
                end for
                Efface le mouvement de la reine sur le plateau
                Remet les positions de la reine comme avant, en ajoutant la position initiale à
la fin de la liste des positions et en retirant l'autre position
            end for
        end for
    else
        meilleurscore = 1000
        meilleurcoup = []
        for reine dans position du joueur blanc do
            for mouvement de reine dans mouvements possibles do
                Bouge la reine sur le plateau
                Modifie la position des joueurs
                for fleche dans mouvements de fleche possibles do
                    Met la flèche sur le plateau
                    score = minimax(plateau, positions_joueur, profondeur - 1, TRUE)[1]
                    coup = position reine départ > mouvement reine > fleche
                    if meilleurscore > score then
                        meilleurscore = score
                        meilleurcoup = [(coup,score)]
                    else if meilleurscore == score then
                        Ajoute le coup dans les meilleurs coups
                    end if
```

Efface la flèche sur le plateau
**end for**
Efface le mouvement de la reine sur le plateau
Remet les positions de la reine comme avant, en ajoutant la position initiale à la fin de la liste des positions et en retirant l'autre position
**end for**
**end for**
**end if**
**return** Un des meilleurs mouvements pris au hasard
**end function**

Le détail de l'implémentation se trouve à la section 2.4.

## 2.3 Fonctions à modifier

Nous vous demandons de modifier deux fonctions de la partie 1 :

— `construire_plateau(n : int, positions_noir : liste de str, positions_blanc : liste de str, positions_fleches : liste de str) -> liste de listes de str` : Vous pouvez maintenant ajouter des flèches à votre plateau lors de sa construction, selon les positions données par `positions_fleches`.

— `main() -> int` : La fonction principale ne reçoit plus de paramètres de fonction, mais commencent plutôt la construction du plateau à partir des paramètres récoltés dans le fichier donné en ligne de commande. Vous devez également adapter votre fonction pour que l'IA soit le joueur noir et joue le coup obtenu avec la fonction `minimax`.

## 2.4 Fonctions à implémenter

Nous vous demandons d'implémenter deux fonctions :

— `lire_fichier(nom : str) -> (int, liste de str, liste de str, liste vide ou liste de str)` : Cette fonction lit le fichier contenant les informations pour le plateau, tel que décrit dans la section 2.1. La fonction renvoie un tuple contenant dans l'ordre : la taille du plateau, les positions des reines noires, les positions des reines blanches et les positions des flèches. La dernière ligne du fichier peut être vide, donc il est possible que vous deviez renvoyer une liste vide pour les positions des flèches.

— `minimax(plateau : liste de listes de str, positions_blanc : liste de str, positions_noir : liste de str, profondeur : int = 2, maxi: bool = True) -> (str ou None, int)` : Cette fonction implémente l'algorithme de minimax décrit un peu plus haut. On commence toujours par une profondeur valant 2, ce qui signifie que l'on va étudier un coup de l'AI, suivant d'une simulation d'un coup humain, et renvoyer le coup et le score attaché au plateau obtenu après ces deux coups. La fonction renvoie un tuple contenant en premier le string du meilleur coup pour le joueur (ou None) et le score de type entier rattaché à ce coup (ou au dernier coup joué). C'est un algorithme récursif, ce qui signifie que la fonction s'appelle elle-même en modifiant les paramètres.

Lorsqu'il y a plusieurs coups avec le même meilleur score, la fonction va renvoyer un des tuples (coup, score) au hasard grâce à la fonction `choice` de la librairie `random`.

Il est très important que vous exploriez la liste des positions des reines dans l'ordre, que vous utilisiez la fonction `mouvements_possibles` que vous avez implémentée en partie 1 pour obtenir les mouvements possibles triés alphabétiquement à explorer également dans l'ordre (pour les mouvements des reines, comme pour les lancers de flèches). Il est également important que vous modifiez les listes de positions en retirant l'ancienne position et en rajoutant la nouvelle position à la fin. Cela est aussi le cas lorsque vous voulez revenir en arrière après avoir testé tous les coups pour un mouvement de reine, vous devez retirer la position à laquelle la reine se trouve de la liste et ajouté sa position initiale en fin de liste.

Evitez les prints (messages d'erreur et autres) pendant l'exécution de la fonction minimax, cela dérangerait votre expérience du jeu. Il est aussi normal que l'IA ne performe pas bien et prenne du temps à prendre les décisions, cela est dû à la fonction de score approximative et le test de tous les coups possibles. Vous explorerez lors des parties futures des façons d'améliorer votre IA.

## 2.5 Librairies permises

— `random`
— `sys`
— `os`

Aucune autre librairie ne sera autorisée.

## 2.6 Consignes de remise

Tout d'abord, insistons sur l'importance de respecter toutes les consignes ci-dessus et ci-dessous : **tout manquement aux consignes sera sanctionné d'une note nulle.**

Afin de créer votre repository pour votre partie 2 sur GitHub Classroom, vous devrez utiliser le lien suivant : `https://classroom.github.com/a/eOS8nfZX`

Voici ensuite les étapes à suivre :

1. Attendez quelques minutes avant de rafraîchir la page pour obtenir le lien vers votre projet. Une fois obtenu, assurez-vous d'avoir bien les dossiers : .gihtub, ressources, source et tests.
2. **Forkez** le repository sur votre compte personnel (voir Figure 2). Cela signifie faire une copie du projet que vous pouvez modifier autant que vous le voulez.
3. Sur **votre repository personnel**, que vous pouvez reconnaître avec car votre github id précède le nom de votre repository et plus `ulb-infof106` comme sur la figure, cliquez sur l'onglet `Actions` et autorisez les `Workflows`. Cela vous permettra d'avoir les tests automatiques sur votre repository, afin de voir les points que vous obtiendrez, que vous trouverez après chaque push dans l'onglet `Actions`.
4. À présent, vous pouvez cloner ce repository et faire tous les commits et push dessus.
5. **AVANT LA DEADLINE**, n'oubliez pas de faire **UN** pull request (onglet sur le repository sur le web) pour transférer votre projet sur l'assignment. Nous ne regarderons que ce qui se trouve sur votre repository dans l'organisation `ulb-infof106`.

L'entièreté de votre code doit se trouver dans un fichier `partie2.py` (avec une première lettre minuscule). **Votre nom, numéro de matricule et section doivent être indiqués en commentaire au début du fichier `partie2.py`.**

Le jeu doit commencer en lançant la commande `python3 -m source.partie2 nom_de_fichier` depuis le dossier GitHub, donc celui où vous voyez les dossiers source et

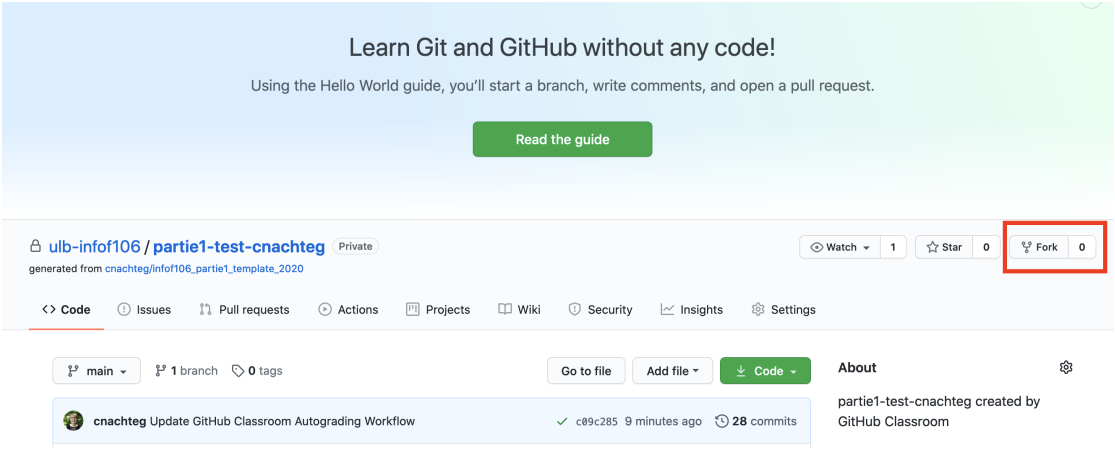



FIGURE 2 – Où trouver le bouton fork sur votre repository GitHub

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

À travers l'entièreté de votre projet, veillez à respecter à la lettre la signature des fonctions (nom + paramètres + type de la valeur retournée) ainsi que la casse des caractères (respect des majuscules et minuscules).

**Uploadez régulièrement votre code** sur le serveur ! Une bonne habitude est de le faire systématiquement à la fin de chaque session de travail, par exemple :

```
git commit -am 'courte description des modifications'
git push
```

La bonne utilisation de git fait partie de la note de ce projet.

Echéance pour la remise du fichier `partie2.py` sur GitHub Classroom dans le dossier source : **dimanche 13 décembre à 22h**.

**Remarques importantes :**

— Il n'y a aucun retard possible, GitHub Classroom n'acceptera aucun `commit` passé cette heure. La version de votre code qui sera évaluée sera la dernière version sur GitHub Classroom.

— Si vous rencontrez des problèmes avec l'utilisation de git, nous vous invitons à relire les slides d'introduction à git disponibles sur l'UV, et à poser vos éventuelles questions qui subsisteraient aux guidances ou aux assistants. Dans tous les cas, posez vos questions **bien à l'avance**, pas deux-trois jours avant l'échéance. Pour rappel, l'usage de git sera évalué, et en particulier la régularité des `commit` et `push`.

— Chaque année, quelques étudiants attendent le dernier moment pour apprendre à utiliser git, et rencontrent des problèmes avec git le weekend de la remise. Il est inutile d'envoyer le projet par email et d'expliquer les problèmes rencontrés, seuls les projets remis via GitHub Classroom sur l'organisation `ulb-infof106` seront évalués.

— Faites bien attention à utiliser le lien GitHub Classroom ci-dessus pour créer votre repository et à forker celui-ci, et à ne pas créer vous-même un repository GitHub standard de votre côté. Ces derniers sont publics (!) par défaut, vous partageriez donc votre code

avec tous les internautes, ce qui serait sanctionné d'une note nulle. Nous insistons sur ce dernier aspect car quelques cas se sont présentés l'année passée lors de la partie 1.

— Ne touchez pas au dossier `tests` du repository ! Nous allons vérifier l'intégrité des fichiers tests après la remise et s'il s'avère que vous les avez modifiés pour réussir les tests, vous aurez une note nulle.

— Nous allons effectuer des tests de plagiat sur vos codes. Si nous détectons du plagiat entre plusieurs personnes, il y a de très grandes chances que tout le monde reçoive une note nulle, et pas seulement la personne qui a copiée le code. Soyez vigilants lorsque vous discutez du projet d'année avec vos collègues.

**Bon travail !**

**Personne de contact : Charlotte Nachtegael – charlotte.nachtegael@ulb.be – N8.213**

# 3 Partie 3 : Restructuration du code et détection anticipée de fin de partie

Cette troisième partie contient deux sous-parties : (i) la restructuration de votre code en classes avec utilisation d'exceptions et (ii) la détection de fin de partie avant qu'il ne reste plus aucun mouvement disponible.

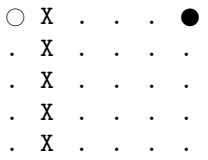
## 3.1 Restructuration du code — 8 points

Vous allez devoir séparer votre code en différentes classes en utilisant plusieurs fichiers et en gérant toutes les erreurs à l'aide d'exceptions. Faites bien attention à toutes les bonnes pratiques vues au cours de programmation, et celles vues au début du cours d'algorithmique (premier chapitre sur les ADTs). En particulier veillez bien à l'encapsulation ainsi qu'à découper votre code sémantiquement.

L'évaluation portera sur la pertinence de la découpe, la qualité du code, ainsi que le respect des bonnes pratiques. Cette restructuration est importante pour une bonne réalisation de la dernière partie !

## 3.2 Détection de fin de partie — 12 points

Actuellement, vous détectez la fin d'une partie lorsqu'un des deux joueurs n'a plus aucun mouvement disponible. Il est cependant possible de terminer une partie de manière anticipée. En effet, il est assez clair dans la disposition suivante que le joueur blanc ne peut pas gagner car il ne peut – au mieux – jouer que 4 tours alors que le joueur noir peut en jouer 19 :



Cette observation vient du fait que si deux reines se situent dans deux régions différentes sans chemin allant de l'une à l'autre, il leur est impossible de se bloquer mutuellement. Il vous faut alors pouvoir identifier de telles régions (appelées *composantes connexes*).

### 3.2.1 Identification des composantes connexes

Une approche classique pour déterminer les composantes connexes est la suivante : si `grid` est la matrice du tableau de jeu, dans une matrice `labels` de même dimension, nous allons stocker un identifiant unique pour chaque composante. Il suffit de parcourir récursivement toutes les directions en partant d'un point donné pour identifier une composante connexe. En effet, par définition, seuls les points de la même composante connexe seront atteints lors de ce parcours.

```
1: function LABEL_ALL_COMPONENTS(grid)
2:     labels ← empty grid N × N
3:     id ← 0
4:     for x = 0 to N − 1, y = 0 to N − 1 do
5:         if grid[y,x] ≠ ARROW and labels[y,x] = 0 then
6:             id ← id+1
7:             LABEL_COMPONENT(grid, id, labels, x, y)
8:         end if
9:     end for
10: end function
11:
12: function LABEL_COMPONENT(grid, id, labels, x, y)
13:     labels[y,x] ← id
14:     for direction (Δx, Δy) do
15:         x′ ← x + Δx
16:         y′ ← y + Δy
17:         if grid[y′,x′] ≠ ARROW and labels[y′,x′] = 0 then
18:             LABEL_COMPONENT(grid, id, labels, x′, y′)
19:         end if
20:     end for
21: end function
```

Attention cependant : identifier les composantes connexes n'est pas suffisant pour déterminer le nombre de coups restants de chaque joueur. En effet, il existe des régions (appelées *défectives*) où le nombre de coups possibles ne correspond pas au nombre de cases libres. Par exemple dans la situation suivante :

```
X X X X X
X X ○ X X
X . X . X
X X X X X
```

bien qu'il y ait deux cases libres, le joueur blanc ne peut jouer qu'un seul tour : soit il part à gauche et tire sa flèche au milieu, soit il part à droite et tire sa flèche au milieu. Dans les deux cas, après ce coup, le joueur blanc est coincé.

On peut tout de même identifier certaines familles de régions que l'on sait non-défectives [3] :

— les lignes ;
— les rectangles ;
— les triangles.

**Lignes** Qu'elles soient horizontales, verticales ou diagonales, les lignes ne sont pas des régions défectives. Il est aisé de s'en convaincre puisque la reine présente dans la ligne peut se placer à l'avant dernière position, tirer sa flèche à la dernière position et recommencer jusqu'à ce qu'elle soit coincée.

**Rectangles** De la même manière, dans un rectangle, la reine présente peut aller dans la dernière ligne, tirer dans le coin inférieur droit et ensuite suivre un chemin l'amenant dans le coin supérieur gauche afin de remplir l'entièreté du rectangle.

3. De manière générale, déterminer si une région du plateau est défective ou non est un problème particulièrement difficile à résoudre. Voilà pourquoi nous nous restreignons à ces sous-familles de régions.

**Triangles** Distinguons deux familles de triangles :

— un triangle de type 1 est un ensemble $T$ de poins tel qu'il existe $A = (x_A, y_A) \in T$ et $n \in \mathbb{N}$ tels que :

$$T = \{(x, y) \text{ s.t. } x \geq x_A, y \geq y_A \text{ et } y - y_A + x - x_A \leq n\},$$

ou une rotation d'un tel ensemble d'un angle de 90°, 180°, ou 270° ;

— un triangle de type 2 est un ensemble $T$ de points tel qu'il existe $A = (x_A, y_A) \in T$ et $n \in \mathbb{N}$ tels que :

$$T = \{(x, y) \text{ s.t. } |x - x_A| \leq n, y - y_A \leq n \text{ et } |x - x_A| + y - y_A \leq n\},$$

ou une rotation d'un tel ensemble d'un angle de 90°, 180°, ou 270°.

Dans l'exemple ci-dessous, les triangles de type 1 sont marqués par des ronds rouges alors que les triangles de type 2 sont marqués par des ronds bleus :

```
8  o o X X o o o o
7  o X X o X o o o
6  X o X o o X o o
5  o o X o X X X o
4  X o X X X o X X
3  o X X X o o o X
2  o o X o o o o o
1  o o o X X X X o
   a b c d e f g h
```

Par triangle nous entendons un ensemble $T$ qui est soit un triangle de type 1, soit un triangle de type 2.

Dans un triangle, la reine présente peut aller sur un des côtés et suivre un chemin *en spirale* afin de remplir l'entièreté de la région.

Dans chacun de ces trois cas ci-dessus, une région de taille $n$ contenant une unique reine permet $n - 1$ coups consécutifs.

Lors d'une partie, si toutes les reines sont isolées dans des composantes connexes différentes d'une des trois formes ci-dessus, il est possible de déterminer qui gagne (en supposant que les deux joueurs jouent de manière *optimale*, *i.e.* en jouant autant de mouvements que possible). De plus, on peut déterminer le score de chacun des joueurs en comptant le nombre de tours où chaque joueur dispose d'un mouvement faisable. Dans l'exemple ci-dessous :

```
8  ○ . X . . . ● .
7  . . X X X X X X
6  X X X . . . . .
5  . X X ○ . . . .
4  . ● X X X X X X
3  . X X . X . X X
2  X X X . X X . X
1  ○ X X . X X X ●
   a b c d e f g h
```

le joueur ○ a une reine dans chacune des trois régions suivantes :

1. le carré (donc rectangle) (`a8`, `b7`) ;
2. le rectangle (`d6`, `h5`) ;

3. la position `a1`,

comptabilisant un total de $3 + 9 + 0 = 12$ points alors que le joueur ● a une reine dans chacune des trois régions suivantes :

1. la ligne (`d8`, `h8`) ;
2. le triangle (`a5`, `b4`, `a3`) ;
3. la ligne (`f3`, `h1`),

comptabilisant un total de $4 + 3 + 2 = 9$ points. Le score est donc de 12 vs 9 pour ○, i.e. ce dernier est gagnant de la partie.

**Remarque :** si vous voulez ajouter certaines formes de composantes connexes non-défectives, vous êtes bien entendu libres de le faire. Pensez alors à l'expliciter dans votre code à l'aide de commentaires. Vous devrez également le justifier dans votre rapport à la fin du projet.

### 3.2.2 Enveloppe convexe

Bien que nous vous laissions le champ libre sur la manière de détecter la forme d'une composante connexe du plateau, nous vous proposons d'utiliser l'algorithme `quickhull` présenté ci-dessous. Cet algorithme permet de déterminer l'*enveloppe convexe* d'un ensemble de points. À vous de voir comment cet algorithme peut vous aider à déterminer les formes décrites dans la section précédente. Voici quelques définitions pour commencer.

Un ensemble $E \subseteq \mathbb{R}^d$ est *convexe* lorsque pour toute paire de points $x, y \in E$, le segment $[x, y]$ est contenu dans $E$ (intuitivement $E$ est convexe si on peut joindre tout point $y$ en partant de tout point $x$ *via une ligne droite*). L'*enveloppe convexe* d'un ensemble $E$ (notée $\mathrm{Conv}\, E$) est le plus petit convexe qui contient $E$. Notons que si $E$ est un ensemble fini de points, alors $\mathrm{Conv}\, E$ est un polygone convexe.

Par exemple, pour $E = \{(x, y) \in \mathbb{N}^2 \text{ s.t. } 0 \leq x \leq 5, 0 \leq y \leq 3\}$, l'enveloppe convexe $\mathrm{Conv}\, E$ est le rectangle défini par les points $\{(0, 0), (5, 0), (5, 3), (0, 3)\}$ (*cf.* Figure 3).

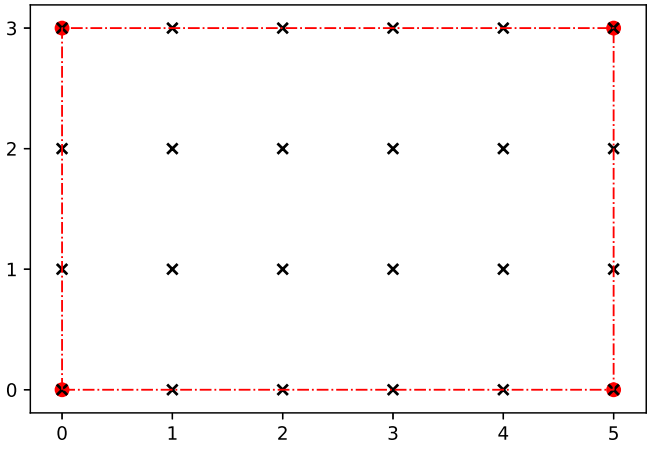



FIGURE 3 – Enveloppe convexe (en trait rouge pointillé) de l'ensemble $E$ des croix noires. Les cercles rouges déterminent les coins de $\mathrm{Conv}\, E$.

L'algorithme `quickhull` est présenté ci-dessous : la fonction `quickhull` prend en paramètre un ensemble de points $S \subseteq \mathbb{R}^2$ et renvoie un sous-ensemble de $C \subseteq S$ contenant les coins de Conv $S$.

```
1: function QUICKHULL(S)
2:     Soient a, b ∈ S avec respectivement la plus petite et la plus grande abscisse
3:     Le segment [a, b] sépare les points de S en deux sous-ensembles disjoints S_0 et S_1
4:     C ← {a, b}
5:     FIND_HULL(S_0, a, b, C)
6:     FIND_HULL(S_1, b, a, C)
7:     return C
8: end function
9:
10: function FIND_HULL(S, a, b, C)
11:     if S = ∅ then
12:         return
13:     end if
14:     Soit c ∈ S le point le plus éloigné du segment ab
15:     C ← C ∪ {c}
16:     Soient S_0, S_1, S_2 la séparation de S décrite ci-dessous
17:     FIND_HULL(S_1, a, c, C)
18:     FIND_HULL(S_2, c, b, C)
19: end function
```

À la ligne 3 de l'algorithme, l'ensemble $S$ est séparé en deux sous-ensembles $S_0$ et $S_1$ : $S_0$ est l'ensemble des points de $S$ à gauche du segment $[a, b]$ et $S_1$ est l'ensemble des points de $S$ à droite du segment $[a, b]$. Remarquez que les points qui sont exactement sur le segment $[a, b]$ sont ignorés (car ils ne donnent aucune information sur l'enveloppe convexe).

**Attention :** le passage de $C$ en paramètre de FIND_HULL est fait par référence et non par copie ! La ligne 15 représente donc la mise à jour de l'ensemble $C$, et non une réassignation.

À la ligne 16 de l'algorithme, l'ensemble $S$ est séparé en trois sous-ensembles : $S_0, S_1$ et $S_2$. Cette séparation se fait comme suit : puisque, par construction, tous les points de $S$ sont du même côté du segment $[a, b]$ et $c$ est le point le plus éloigné du segment $[a, b]$, le triangle $abc$ permet de séparer $S$ en les trois sous-ensembles disjoints suivants (*cf.* Figure 4) :

— les points à l'intérieur du triangle ($S_0$) ;
— les points à gauche du segment $[a, c]$ ($S_1$) ;
— les points à droite du segment $[b, c]$ ($S_2$).

Puisque les points $a, b$ et $c$ font partie de l'enveloppe convexe de l'ensemble de départ, il est évident que le sous-ensemble $S_0$ ne nous intéresse pas puisqu'aucun des points qu'il contient de peuvent faire partie de Conv $E$. Tant que les régions $S_1$ et $S_2$ ne sont pas vide, on sait qu'il est possible de rajouter des points à Conv $E$.

## 3.3 Librairies permises

Vous avez droit à tous les packages standards de Python 3 (*cf.* `https://docs.python.org/3/library/index.html`) mais attention à les utiliser correctement !

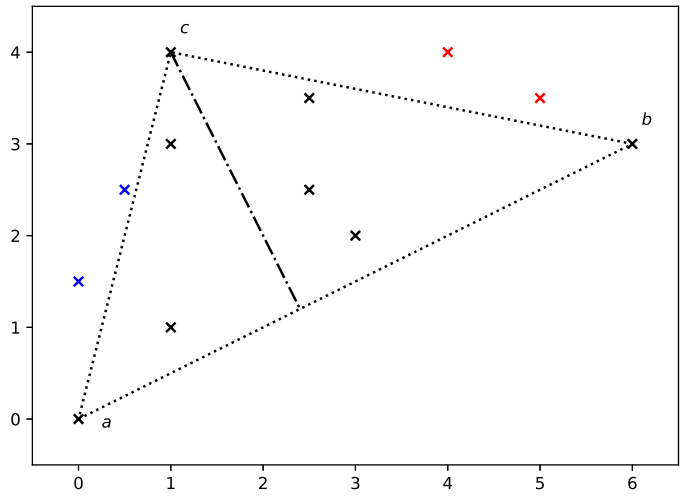



FIGURE 4 – Les sous-ensembles $S_0, S_1$ et $S_2$ sont marqués respectivement en noir, bleu et rouge.

## 3.4 Consignes de remise

Rappelons une fois de plus que **tout manquement aux consignes sera sanctionné d'une note nulle**.

Afin de créer votre repository pour votre partie 3 sur GitHub Classroom, vous devrez utiliser le lien suivant : `https://classroom.github.com/a/zuIPW2vP`.

Votre code doit se lancer via la commande `python3 partie3.py <chemin>` (tout en minuscule, sans espace, sans underscore, *etc.*) où `<chemin>` est le chemin vers un fichier respectant le format demandé dans la partie 2. Il faut que le fichier `partie3.py` se situe directement à la racine du repo, vous pouvez par contre mettre le reste des fichiers dans un ou plusieurs sous-dossier(s) si vous le souhaitez. **Ce même fichier doit commencer par le commentaire suivant :**

```
"""
Nom:
Prénom:
Matricule:
Section:
"""
```

**Uploadez régulièrement votre code** sur le serveur ! Une bonne habitude est de le faire systématiquement à la fin de chaque session de travail, par exemple :

```
git status              # pour voir les fichiers modifiés
git add *.py            # pour ajouter tous les fichiers source modifiés
git commit -m "courte description des modifications"
git push origin master  # pour envoyer sur GitHub
```

Échéance pour la remise sur GitHub Classroom dans le dossier source : **dimanche 28 février à 22h**.

**Remarques importantes :**

- Il n'y a aucun retard possible, GitHub Classroom n'acceptera aucun `commit` passé cette heure. La version de votre code qui sera évaluée sera la dernière version sur GitHub Classroom, sur la branche `main` (ou `master`).
- Si vous rencontrez des problèmes avec l'utilisation de git, nous vous invitons à relire les slides d'introduction à `git` disponibles sur l'UV, et à poser vos éventuelles questions qui subsisteraient aux guidances ou aux assistants. Dans tous les cas, posez vos questions **bien à l'avance**, pas deux-trois jours avant l'échéance.
- Il est inutile d'envoyer le projet par email et d'expliquer les problèmes rencontrés, seuls les projets remis via GitHub Classroom sur l'organisation `ulb-infof106` seront évalués.
- Faites bien attention à utiliser le lien GitHub Classroom ci-dessus pour créer votre repository, et à ne pas créer vous-même un repository GitHub standard de votre côté. Ces derniers sont publics (!) par défaut, vous partageriez donc votre code avec tous les internautes, ce qui serait sanctionné d'une note nulle. Nous insistons sur ce dernier aspect car quelques cas se sont présentés l'année passée lors de la partie 1.
- Nous allons effectuer des tests de plagiat sur vos codes. Si nous détectons du plagiat entre plusieurs personnes, il y a de très grandes chances que tout le monde reçoive une note nulle, et pas seulement la personne qui a copiée le code. Soyez vigilants lorsque vous discutez du projet d'année avec vos collègues.
- Pensez à bien lire la FAQ avant de poser vos éventuelles questions : il est possible que la réponse s'y trouve déjà.

**Bon travail !**

**Personne de contact : Robin Petit – robin.petit@ulb.be – O8.210**

# 4 Partie 4 : Interface graphique et amélioration de l'IA

La quatrième partie du projet d'année consiste principalement à réaliser une *interface graphique* (***G**raphical **U**user **I**nterface*) de votre programme et à améliorer l'IA que vous avez développée lors des phases précédentes grâce à une ou plusieurs fonctions heuristiques. 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` [4]. 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.

Pour cette partie, vous devrez implémenter une série de fonctionnalités obligatoires ainsi que choisir entre différentes fonctionnalités supplémentaires. Vous êtes bien évidemment encouragés à implémenter plusieurs de ces propositions si vous avez le temps et l'envie.

Outre la partie implémentation, vous pourrez perfectionner votre IA afin de participer à une compétition entre IAs lors de laquelle vous aurez l'occasion de sauver le monde et d'obtenir des points bonus.

## 4.1 Interface graphique (12 points)

### 4.1.1 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 ne vous est pas autorisée et entraînera une note nulle** ; il vous faut donc aller lire la documentation et créer vous-même vos layouts et votre arborescence de widgets. Vous aurez l'occasion d'utiliser Qt Designer lors de futurs projets durant vos études ; l'objectif ici est de faire la GUI "à la main" afin de bien comprendre son fonctionnement.

### 4.1.2 Exemple de maquette et éléments requis

Vous trouverez une proposition de maquette pour votre programme sur la Figure 5. Cette dernière est découpée en 4 rectangles rouges contenant respectivement :

4. Tutorial : `https://pythonspot.com/pyqt5/`,
documentation : `https://www.riverbankcomputing.com/static/Docs/PyQt5/`.

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.

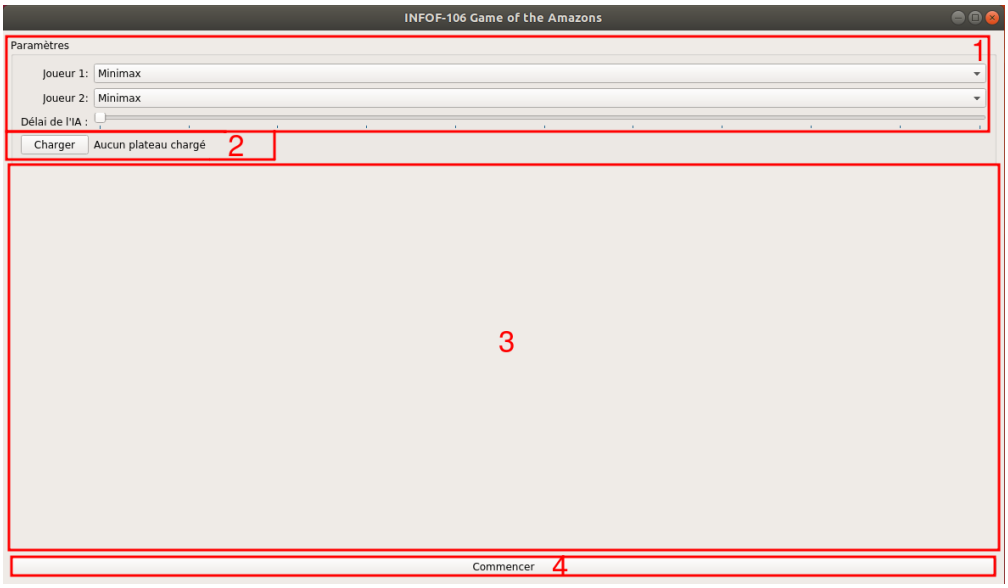



FIGURE 5 – Exemple de maquette.

La Figure 6 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 élements 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.

### 4.1.3 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 [5] ;
— un dessin du plateau de jeu représentant les cases occupées par les reines et les flèches 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 ;

5. Si les joueurs 1 et 2 sont tous deux des joueurs IA utilisant l'algorithme minimax avec une faible profondeur, chaque décision est prise en quelques dixièmes de seconde, dès lors la partie se déroulerait trop vite, et on ne la verrait pas se dérouler sur l'interface.

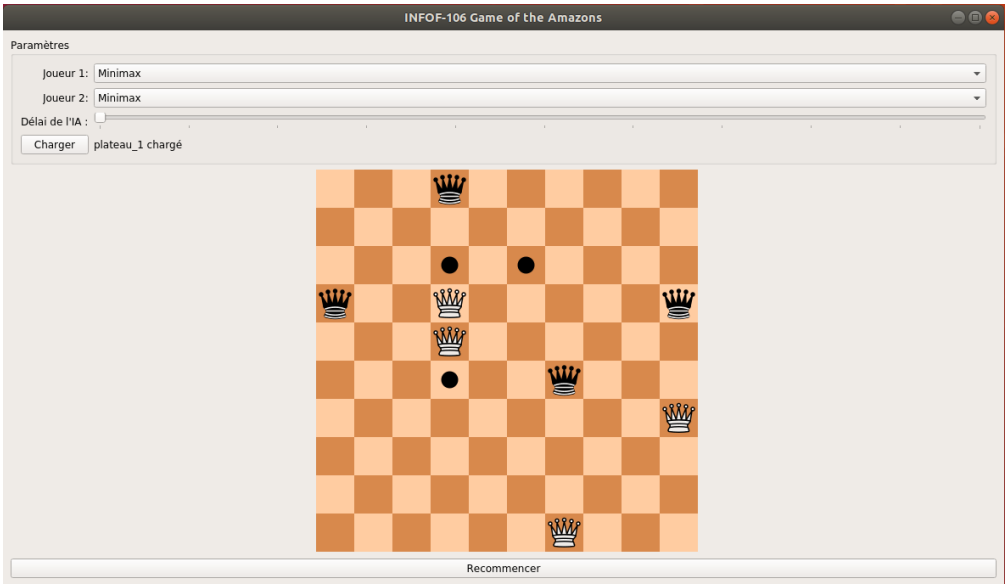



FIGURE 6 – Exemple de maquette pendant le déroulement d'une partie.

Lorsqu'un joueur humain désire jouer un coup, il doit cliquer sur l'une de ses reines (dont la sélection doit être visible graphiquement), ensuite cliquer sur une case vide valide sur laquelle la reine sélectionnée se déplacera et enfin cliquer sur une case vide valide à laquelle la reine tirera une flèche.
Le temps pris entre deux coups de l'IA peut être modifié mais les joueurs eux ne peuvent pas l'être. 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.

### 4.1.4 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 `partie4.py` à l'aide du mot-clef `import`.

### 4.1.5 Dessin sur un canevas `PyQt5`

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QPushButton, QMessageBox
from PyQt5.QtGui import QIcon
from PyQt5.QtCore import pyqtSlot


class App(QWidget):

    def __init__(self):
        super().__init__()
        self.title = 'PyQt5 messagebox - pythonspot.com'
        self.left = 10
        self.top = 10
        self.width = 320
        self.height = 200
        self.initUI()

    def initUI(self):
        self.setWindowTitle(self.title)
        self.setGeometry(self.left, self.top, self.width, self.height)

        buttonReply = QMessageBox.question(self, 'PyQt5 message', "Do you like PyQt5?",
                                           QMessageBox.Yes | QMessageBox.No, QMessageBox.No)
        if buttonReply == QMessageBox.Yes:
            print('Yes clicked.')
        else:
            print('No clicked.')

        self.show()


if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = App()
    sys.exit(app.exec_())
```

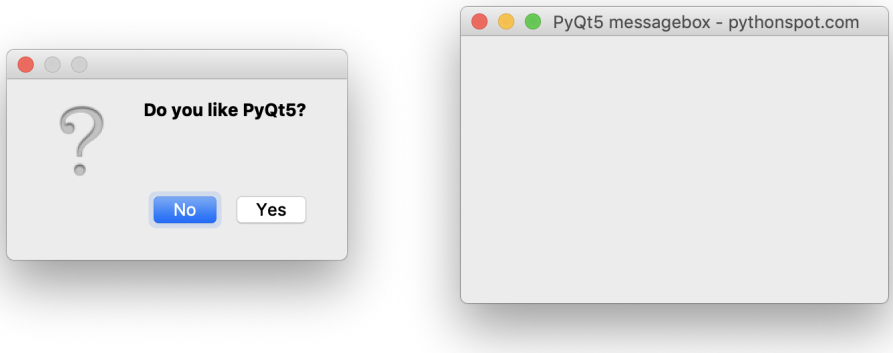



FIGURE 7 – Exemple de maquette *PyQt5*.

La Figure 7 donne un exemple de code `PyQt5` créant une fenêtre de message qui reçoit un signal et fait une action en fonction de la réponse.

De la même manière, on peut ajouter des objets de types `Button`, `Label`, `Menu`, `Text` (et beaucoup d'autres) sur une `Window`. Nous vous invitons à explorer les différents widgets qui existent dans `PyQt5` afin de choisir les éléments qui conviennent le mieux à votre interface. N'hésitez pas à utiliser des couleurs dans votre interface graphique, tant qu'elle reste lisible.

Vous trouverez un tutoriel sur les différents widgets à l'adresse suivante : `https://pythonspot.com/pyqt5/`.

## 4.2 Participation au grand tournoi Kombat of the Amazons (5 points)

L'empereur de l'Outre-Monde, Gwenaël Joret, a envoyé ses plus grands guerriers (vos assistants) conquérir le Royaume de la Terre. Seulement, les Dieux Anciens avaient décrété qu'un royaume ne pouvait en envahir un autre qu'après avoir battu son champion au **grand tournoi Kombat of the Amazons**.

Nous allons organiser une **grande compétition entre IAs dont l'issue déterminera le sort du Royaume de la Terre**. Puisque tant de vies dépendent de votre succès, la participation au grand tournoi est **obligatoire**. Outre l'immense honneur de sauver votre royaume (et vous-même) de l'extinction, les 16 premiers recevront également des points bonus, voir ci-dessous.

Afin de participer au grand tournoi Kombat of the Amazons, vous devez ajouter dans votre repository un fichier `tournoi.py` qui contiendra une classe nomée `TournoiAIPlayer` contenant l'implémentation de votre meilleure IA qui vous représentera dans le tournoi. Votre classe contiendra également un docstring détaillant toutes les améliorations effectuées à votre IA. Concrètement, le squelette de votre classe est le même que celui de la classe `AIPlayer` proposé dans la correction de la phase 3, c'est-à-dire :

```
from players import Player, AIPlayer


class TournoiAIPlayer(AIPlayer):
    '''
    Your best IA that will represent you during the tournament.
    '''
    def __init__(self, board, player_id):
        super().__init__(board, player_id)

    def play(self):
        '''
        Play the best action.
        '''
        # Call the method that will return the best action
        # ...
```

Le code de l'IA qui participera au tournoi doit entièrement être encapsulé dans la classe `TournoiAIPlayer`, seul votre fichier `tournoi.py` sera importé durant le grand tournoi. Si vous ne respectez pas le nom du fichier ou de la classe, ou que votre code n'est pas fonctionnel, cela sera considéré comme un K.O. technique (TKO). Dans ce cas, vous ne serez pas en mesure de participer au grand tournoi Kombat of the Amazons et, outre l'immense honte que resentiront vos disciples, vous ne recevrez pas les points pour cette partie.

**Précisions d'implémentation importantes :** la classe TournoiAIPlayer contenue dans tournoi.py doit pouvoir être importée depuis n'importe quel autre projet. Cette classe doit recevoir les mêmes paramètres d'initialisation que dans la correction de la partie 3 et contenir les mêmes méthodes. Ce seront ces méthodes que nous appellerons lors du tournoi. Le plateau de jeu doit être une classe qui implémente les mêmes méthodes (notamment la méthode `act()`) que dans la correction de la partie 3. Concrètement, pour le tournoi, nous allons simplement importer votre classe TournoiAIPlayer, l'instancier et appeler la méthode play. Par contre, il est important que votre méthode play appelle la méthode `act` de la classe Board telle qu'elle est définie dans la correction de la partie 3. **Concrètement, si vous voulez tester que votre TournoiAIPlayer fonctionnera lors du tournoi, vous pouvez utiliser la correction de la partie 3 en remplaçant la classe AIPlayer par votre nouvelle IA.**

### 4.2.1 Minuterie

Afin de mettre toutes les IA sur le même pied d'égalité pour le grand tournoi, nous vous demandons d'**implémenter une minuterie** qui limitera le temps que pourra prendre l'IA pour retourner son prochain coup. Vous devrez également gérer le cas où une IA ne respecte pas la minuterie en disqualifiant immédiatement l'IA. Concrètement, si une IA ne retourne pas un coup dans le temps imparti, elle perd la partie. Nous vous demandons de fixer votre minuterie à **2 secondes** par défaut, comme il sera d'usage lors du grand tournoi. Il est évident que cette limitation ne s'applique pas à un joueur humain.

Pour l'implémentation de la minuterie, vous devez utiliser la librairie `time` de Python. **Aucune** autre librairie n'est autorisée pour la gestion du temps.

### 4.2.2 Approfondissement itératif

Un nouveau facteur important à prendre en compte dans cette phase 4 est donc l'introduction d'une minuterie qui limite à **2 secondes** le temps que pourra prendre votre IA pour retourner son prochain coup. Un effet, le non-respect de cette règle entraînera une disqualification immédiate du grand tournoi et donc une chance en moins d'empêcher la destruction du Royaume de la Terre. Dans cette situation, il est délicat de choisir à l'avance la profondeur de l'arbre du Minimax. En effet, une profondeur **trop grande** au début de la partie vous fera **risquer l'élimination** tandis qu'une profondeur **trop faible** en fin de partie l**imitera les capacités de votre IA**. Heureusement, il existe une technique simple afin d'éviter l'élimination sans trop altérer les capacités de votre IA : **l'approfondissement itératif** (en anglais *iterative deepening*).

Imaginons que vous souhaîteriez utiliser une profondeur maximale $d$ pour votre algorithme Minimax. Au lieu d'estimer le meilleur coup possible sans savoir si votre algorithme arrivera à trouver une solution dans le temps imparti, vous allez d'abord estimer une solution avec une profondeur de 1, puis de 2, etc.. jusqu'à $d$. À chaque étape, vous allez sauvegarder la meilleure solution. Lors de l'estimation d'une solution, si vous vous rendez compte que vous approchez dangereusement du temps imparti, vous abandonnerez l'estimation en cours et vous retournerez la meilleure solution déjà trouvée.

Pour cette amélioration, vous devez **impérativement** utiliser la librairie `time` de Python.

### 4.2.3 Organisation du tournoi

Ce tournoi sera public, nous l'organiserons après les vacances de printemps (date à préciser). Nous utiliserons un système à élimination directe avec huitième de finale, quart de finale, demi-finale et une finale pour terminer. Lorsque deux IAs sont mises en compétition, chacune

jouera une fois comme joueur 1 et une fois comme joueur 2. Une IA remporte la manche si elle gagne les deux parties. En cas d'égalité, nous recommencerons jusqu'à ce qu'un vainqueur soit désigné.

Les points bonus seront ajoutés à la note totale du projet sur 100 comme suit :

| | |
|---|---|
| $1^{\text{ère}}$ place | +10 points |
| $2^{\text{ème}}$ place | +8 points |
| $3^{\text{ème}}$-$4^{\text{ème}}$ place | +6 points |
| $5^{\text{ème}}$-$8^{\text{ème}}$ place | +4 points |
| $9^{\text{ème}}$-$16^{\text{ème}}$ place | +3 points |
| $17^{\text{ème}}$-$32^{\text{ème}}$ place | +2 points |
| $33^{\text{ème}}$-$64^{\text{ème}}$ place | +1 points |

**L'avenir du Royaume de la Terre est en jeu. Votre destin est entre vos mains !**

## 4.3 Amélioration de l'IA (3 points)

L'algorithme Minimax implémenté durant les phases précédentes est certes un noble combattant mais il ne fait pas le poids face aux IA championnes de l'Outre-Monde. Si vous voulez avoir la moindre chance de gagner le grand tournoi Kombat of the Amazons et sauver le Royaume de la Terre, il vous faudra améliorer votre IA, la rendre plus puissante en utilisant les dernières techniques développées dans votre royaume. Vous êtes bien sûr libre d'implémenter vos propres améliorations afin de présenter votre meilleure IA au grand tournoi.

La technique d'une IA est ici représentée par sa stratégie mise en place afin de choisir le meilleur coup possible et de maximiser la probabilité de remporter la partie. Or, au début de la partie, l'algorithme du Minimax que vous avez implémenté dans les phases précédentes joue de manière aléatoire tant qu'il n'arrive pas à déterminer un scénario de fin d'une partie en utilisant la profondeur de recherche fixée. Cette faille pourrait notamment être exploitée par l'adversaire et lui donner le temps nécessaire pour mettre en place un plateau de jeu plus favorable pour lui.

Afin d'améliorer la technique de l'IA, au lieu de renvoyer la valeur 0 lorsque l'algorithme du Minimax n'a pas trouvé de scénario de fin d'une partie en utilisant la profondeur de recherche fixée, nous allons renvoyer, pour chaque scénario une valeur appelée `heuristique`. Cette valeur correspond à une estimation de la qualité d'un scénario possible (c'est-à-dire vos chances de gagner) estimée en fonction du plateau de jeu atteint à l'extrémité d'une branche du Minimax.

Concrètement, dans votre code du Minimax, si la variable `profondeur` est égale à 0, vous allez retourner une valeur estimée par la méthode `objective_function` comme suit :

```python
class MinimaxPlayer(AIPlayer):
    '''
    Specialized form of AI Player using minimax algorithm.
    '''

    def __init__(self, board, player_id):
        super().__init__(board, player_id)


    def minimax(self, depth=2, maximizing=True):
        if depth == 0:
            return (None, self.objective_function())

        (...)


    def objective_function(self):

        # TODO Compute the objective function
        (...)

        return objective
```

Un exemple de fonction objectif simpliste serait une fonction qui compte le nombre de cases du plateau accessibes par les reines de chaque joueur. Ceci n'est qu'un exemple, il existe une multitude de possibilités pour estimer vos chances de remporter la partie en fonction du tableau de jeu. **C'est à vous à être créatif afin de trouver la ou les meilleurs fonctions objectif qui amélioreront vos IA**. Les grands champions développent leurs propres techniques et la différence de techniques entre vos IA dépendra essentiellement de cette estimation !

## 4.4 Améliorations facultatives

### 4.4.1 Combinaisons de différentes heuristiques (2 points bonus)

C'est bien connu, les grands champions ne se limitent pas à une seule technique mais cherchent à combiner différentes techniques afin d'améliorer leurs performances. Peut-être connaissez-vous déjà le terme de *combo* ? Ici, nous parlerons plutôt de *combinaison linéaire*. En effet, il se peut qu'une combinaison pondérée de techniques (heuristiques) différentes produise un meilleur résultat que l'utilisation d'une seule heuristique.

Si vous avez implémenté au moins deux heuristiques, vous pouvez construire une nouvelle estimation en multipliant chaque heuristique par une constante et en ajoutant le résultat. Concrètement, soit un nombre $n \in \mathbb{N}^+$ d'heuristiques $h_0$ $h_1$ ... $h_n \in \mathbb{R}$ estimées par vos fonctions *objectif* et de constantes $c_0$ $c_1$ ... $c_n \in \mathbb{R}$ déterminées, l'heuristique finale est déterminée par la combinaison linéaire suivante :

$$h_{\text{finale}} = c_0 \times h_0 + c_1 \times h_1 + ... + c_n \times h_n$$

À nouveau, soyez créatif afin de développer votre propre combinaison de techniques qui fera assurément la différence lors du grand tournoi Kombat of the Amazons !

#### 4.4.2 Sauvegarde et rediffusion d'une partie (3 points bonus)

Nous vous demandons d'implémenter la possibilité de sauvegarder une partie dans un fichier et de pouvoir rediffuser une partie précédemment jouée. La sauvegarde d'une partie doit être une option (par exemple une case à cocher) que l'on doit pouvoir sélectionner avant le début de la partie. Pour rediffuser une partie, vous devez permettre à l'utilisateur de sélectionner le fichier de la partie qu'il désire revoir. Pour ce faire, nous vous demandons d'utiliser la classe **QFileDialog** de *PyQt*.

Vous avez la liberté d'implémenter ces modifications comme vous le désirez. Veillez cependant à ce que votre interface graphique soit ergonomique, agréable visuellement et simple d'utilisation.

#### 4.4.3 Élagage Alpha-Beta (2 points bonus)

Vous avez développé de nouvelles techniques sans pareil ! Maintenant, il vous reste encore à épurer vos mouvements. L'élagage alpha-bêta (abrégé élagage $\alpha - \beta$) est une technique ancestrale permettant de réduire le nombre de noeuds évalués par l'algorithme minimax en n'évaluant pas les noeuds dont on est sûr que leur qualité sera inférieure à un noeud déjà évalué. En n'explorant qu'une partie de l'arbre, vous pourriez même peut-être augmenter la profondeur de recherche si votre algorithme est assez rapide, à vous de tester.

Une restranscription du parchemin décrivant l'algorithme ancestral de l'élagage $\alpha - \beta$ est disponible à l'adresse `https://fr.wikipedia.org/wiki/%C3%89lagage_alpha-b%C3%AAta`. Le pseudocode décrit n'est peut-être pas exactement celui que vous avez implémenté mais est très similaire. Il vous revient le soin d'adapter votre algorithme Minimax pour y ajouter l'élagage $\alpha - \beta$ en fonction du pseudocode décrit.

#### 4.4.4 Ambiance musicale (1 point bonus)

Que serait un jeu sans ambiance musicale ? Vous pouvez compléter votre interface graphique par une ambiance musicale qui ravira vos joueurs.

Pour cette amélioration, vous devez **impérativement** utiliser la librairie `pygame` de Python.

#### 4.4.5 Autres variantes et ajouts (points à définir selon la proposition)

Vous pouvez, et êtes encouragés, à développer vos propres idées et variantes personnelles. Pour cette partie 4, la créativité est encouragée ! Vous pouvez également implémenter d'autres idées non mentionnées ici provenant de la littérature.

Nous vous demandons alors de contacter l'assistant au préalable afin de vous assurer que votre proposition est adéquate pour ce projet et de connaître sa valorisation en terme de points bonus.

### 4.5 Consignes de remise

Votre code doit se lancer via la commande `python3 partie4.py` (tout en minuscule, sans espace, sans underscore, *etc.*). Il faut que le fichier `partie4.py` se situe directement à la racine du repo Github, vous pouvez par contre mettre le reste des fichiers dans un ou plusieurs sous-dossier(s) si vous le souhaitez. Vous devez structurer vos classes dans différents fichiers. Veillez à ne pas oublier d'indiquer votre **prénom, nom et matricule dans le header de chaque script Python**.

Pour participer au grand tournoi Kombat of the Amazons, vous devez ajouter dans votre repository un fichier `tournoi.py` qui contiendra une classe nomée `TournoiAIPlayer` contenant l'implémentation de votre meilleure IA.

**À remettre :** Les scripts et les autres fichiers nécessaires au fonctionnement du projet (par exemple des images ou autres fichiers texte ou scripts python que vous utiliseriez), dont le fichier `partie4.py`. Notez qu'il n'y a aucun retard possible, la dernière version évaluée sera la dernière version présente sur votre espace GitHub Classroom à la date et heure limite de soumission.

**Échéance de remise sur GitHub Classroom :** Mercredi 7 avril à 22h00.
Lien GitHub Classroom pour la partie 4 : `https://classroom.github.com/a/8voOS-pt`

**Uploadez régulièrement votre code** sur le serveur ! Une bonne habitude est de le faire systématiquement à la fin de chaque session de travail, par exemple :

```
git status              # pour voir les fichiers modifiés
git add *.py            # pour ajouter tous les fichiers source modifiés
git commit -m "courte description des modifications"
git push origin master  # pour envoyer sur GitHub
```

**Personne de contact : Cédric Simar - cedric.simar@ulb.ac.be - N8.212**

# 5 Rapport

Nous vous demandons de rédiger un rapport faisant entre 4 et 6 pages (page de garde, annexes et table des matières non comprises). Ce rapport devra contenir une analyse approfondie ainsi que des graphiques comparatifs des différentes IA implémentées durant le projet d'année, mettant également en évidence l'influence des différents paramètres et motivant vos choix d'implémentation. Le rapport devra obligatoirement être rédigé en LaTeX, en utilisant un template qui vous est fourni sur l'UV.

**Le rapport devra porter uniquement sur les différentes IAs implémentées lors de la partie 4** du projet (celui-ci peut bien sûr évoquer les fonctionnalités de base implémentées aux parties 1, 2, et 3). L'interface graphique et son implémentation ne doivent pas y être décrites.
Ce rapport devra contenir les éléments suivants :

— une page de garde qui reprend vos coordonnées, la date, le titre,... ;
— une table des matières ;
— une introduction et une conclusion ;
— des sections décrivant ce qui a été réalisé, les comparaisons de performance, etc. ;
— des exemples d'exécution de votre code ;
— éventuellement des références bibliographiques si applicable.

Le rapport doit détailler vos choix d'implémentation, les éventuelles difficultés rencontrées, l'analyse et les solutions proposées. Il doit également contenir une description des améliorations de l'IA par rapport à l'IA Minimax de base de la partie 2, ainsi que des comparaisons de performances en fonction des différentes méthodes et valeurs de paramètres utilisés. Nous vous laissons libres de décider vous même de la meilleure manière d'effectuer ces comparaisons, c'est à réflechir, cela fait partie de l'exercice !

**L'utilisation de graphiques est fortement conseillée**. Pour cela, nous vous conseillons l'utilisation de la librairie Python **Matplotlib** que vous devrez également utiliser plus tard dans votre parcours académique. Une brève introduction à cette librairie est accessible à l'adresse `http://math.mad.free.fr/depot/numpy/courbe.html`. Pour plus d'informations, veuillez vous référer à la documentation officielle.

Le rapport doit être clair et compréhensible pour un étudiant imaginaire qui aurait suivi les cours INFO-F-101 et INFO-F-103 mais n'aurait pas fait le projet ; ni trop d'informations, ni trop peu. Expliquez toutes les notions et terminologie que vous introduisez.

Le rapport **doit être écrit en LaTeX**, un système très puissant pour une mise en page de qualité, en utilisant un template qui vous sera fourni ultérieurement. Il est évident qu'une bonne orthographe sera exigée. **Tout rapport écrit dans un autre format (Word ou autre) que celui du template LaTeX fourni ne sera pas corrigé.**
Le rapport sera remis sous format électronique uniquement, pas de remise papier.

## 5.1 Consignes de remise

Votre rapport sera rendu en deux formats, pdf ET LaTeX, dont les fichiers seront nommés respectivement **rapport.pdf** et **rapport.tex**. Tout manquement à cette règle résultera en une note nulle.

**À remettre :** Les fichiers **rapport.pdf** et **rapport.tex**, ainsi que les autres fichiers sources éventuels (images, etc.). Nous devons pouvoir compiler votre code LaTeX à partir de votre repository.

**Deadline sur GitHub Classroom :** Dimanche 9 mai à 22h.
Lien GitHub Classroom pour le rapport : `https://classroom.github.com/a/HhAVl7hK`

Notez qu'il n'y a aucun retard possible. **Uploadez régulièrement votre code** sur le serveur ! Une bonne habitude est de le faire systématiquement à la fin de chaque session de travail, par exemple :

```
git status              # pour voir les fichiers modifiés
git add *.tex *.pdf     # pour ajouter les fichiers LaTeX et pdf modifiés
git commit -m "courte description des modifications"
git push origin main  # pour envoyer sur GitHub
```

**Personne de contact : Gwenaël Joret**