Projet de Programmation C – Gounki

LC4 – Licence 2 – Université Paris Diderot

À rendre pour le 11 mai 2014

Échéances :

— La dernière soumission acceptée doit être envoyée sur DidEL avant le 11 mai à 18h00.
— Les soutenances auront lieu à une date communiquée ultérieurement.

1 Présentation du jeu Gounki

1.1 Vue d'ensemble

Matériel

Gounki se joue à deux sur un échiquier standard, avec des pions blancs d'un côté et noirs de l'autre. Chaque joueur commence avec 16 pions, 8 *ronds* et 8 *carrés*, posés sur les premières lignes de son côté : les ronds sur les cases claires, les carrés sur les cases foncées.

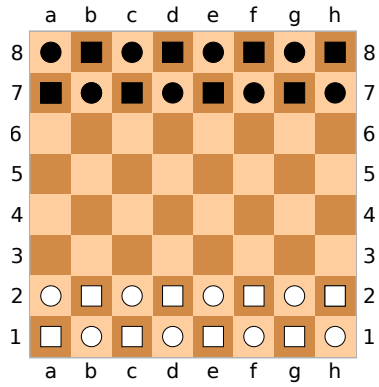


FIGURE 1 – Situation initiale

But du jeu

Le but du jeu est d'être le premier à ramener au moins un de ses pions *au-delà* du bord opposé de l'échiquier, c'est-à-dire au-delà du bord le plus proche du joueur en face.

1.2 Règles

Organisation générale

Chaque joueur joue à tour de rôle une action, qui peut être de deux types : soit un déplacement, soit un déploiement. Comme aux échecs, le joueur blanc commence la partie.

Déplacements de pions simples

Les pions ronds se déplacent en diagonale vers l'avant, d'une case à chaque tour. Les carrés peuvent se déplacer horizontalement à gauche ou à droite, ou verticalement vers l'avant, toujours d'une case par tour. Dans tous les cas, un pion ne peut jamais revenir en arrière.

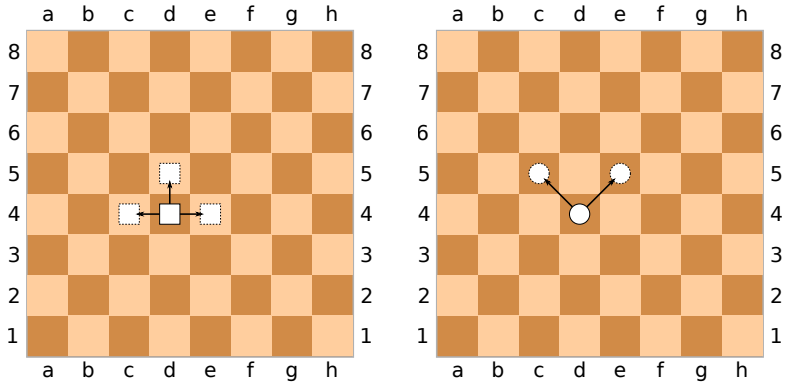


FIGURE 2 – Mouvement des pièces simples

Capture

Quand un pion, simple ou composé, se déplace sur une case occupée par un pion ennemi, que ce dernier soit simple ou composé, il le *capture*. Le pion capturé sera écarté du jeu.

Composition de pions

Un pion peut aussi se déplacer ou se dépiler (voir le *déploiement*), en respectant les contraintes de composition, sur une case occupée par un pion ami, pour se composer avec le pion qui y est déjà. Un pion composé, aussi appelé une *pile* de pions, peut être composé de deux ou trois pions, et pas plus. Comme l'ordre de la composition est indifférent, on peut avoir 7 types de pions composés :

— les double-pions :

1. *double-rond*, composé de deux ronds,
2. *rond-carré*, composé d'un rond et d'un carré,
3. *double-carré*, composé de deux carrés,

— les triple-pions :

4. *triple-rond*, composé de trois ronds,
5. *carré-double-rond*, composé de deux rond et d'un carré,
6. *rond-double-carré*, composé d'un rond et de deux carré, et
7. *triple-carré*, composé de trois carrés.

Donc, un pion simple peut se déplacer sur une case occupée par un pion simple ou un double-pion de la même couleur pour se composer avec celui-ci. Les double-pions peuvent se déplacer sur une case contenant seulement un pion simple de la même couleur. Les compositions plus grandes que trois n'étant pas permises, il n'y a pas d'autre cas de déplacement possible sur les cases contenant un pion ami.

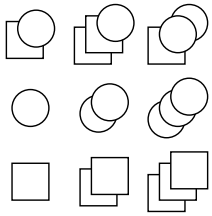


FIGURE 3 – Ensemble des pions existants

Déplacements de pions composés

Un pion composé se déplace en tant qu'une seule unité, et ne peut pas se décomposer tout en se déplaçant (nous verrons plus tard qu'il peut tout de même se décomposer en *se déployant*).

Les possibilités de déplacement d'un pion composé sont déterminées en fonction de celles de ses composants : la portée des pions de la même forme s'ajoute, et les pièces de formes différentes offrent des choix de direction. Un pion composé (comme un pion simple par ailleurs) ne peut pas sauter un autre pion, ami comme ennemi. De plus, un pion composé ne peut pas changer de type de déplacement, ni de direction en cours de mouvement : il doit soit se comporter comme un carré de portée 1, 2 ou 3, soit comme un rond de portée 1, 2 ou 3, selon les pièces qui le composent.

Ainsi, un double-rond (resp. un double-carré) peut se déplacer d'une ou de deux cases diagonalement vers l'avant (resp. verticalement vers l'avant ou horizontalement vers la gauche ou la droite) ; alors qu'un rond-carré peut se déplacer *soit* comme un rond, *soit* comme un carré.

De même, un triple-rond (resp. un triple-carré) peut se déplacer d'une, de deux ou de trois cases diagonalement vers l'avant (resp. verticalement vers l'avant ou horizontalement vers la gauche ou la droite), alors qu'un carré-double-rond (resp. un rond-double-carré) peut se déplacer soit comme un double-rond (resp. un double carré) soit comme un simple carré (resp. un simple rond).

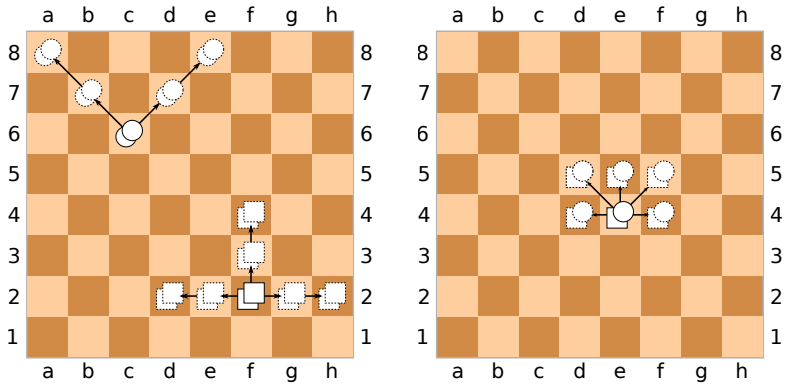


FIGURE 4 – Mouvement des pièces doubles

Dans aucun de ces cas, un pion composé ne peut changer de direction pendant un déplacement. Il y a

un seul cas de changement de direction de mouvement possible, et c'est les *rebonds*.

Déploiement de pions composés

Pour décomposer une pile de pions et séparer ses composants, il faut le déployer. Un pion composé se déploie toujours en pions simples ; c'est-à-dire qu'on ne peut pas décomposer un triple-pion en un pion simple et un double pion. Un déploiement ne prend qu'un seul tour, pendant lequel on le joue au lieu d'un mouvement.

Pour déployer un pion composé de ronds et de carrés, on a le choix de commencer par *dépiler* le(s) rond(s) et passer au(x) carré(s) une fois qu'il n'y a plus de rond, ou bien commencer par le(s) carré(s) et passer au(x) rond(s) une fois qu'il n'y a plus de carré. Il est en revanche interdit de déplier un carré-double-rond, par exemple, en commençant par un rond, passant ensuite par un carré et dépilant finalement le deuxième rond.

Pour dépiler un composant rond (resp. carré) d'un pion composé, on déplace d'abord le pion composé comme un rond (resp. comme un carré), on détache le pion rond (carré) du pion composé, on le laisse à cette case, et on passe au composant suivant, jusqu'à ce qu'il n'y ait plus de composants.

Quand un pion composé comporte plusieurs ronds ou plusieurs carrés, il est obligatoire qu'ils soient tous dépilés dans la même direction et, comme on l'a déjà dit, à la suite.

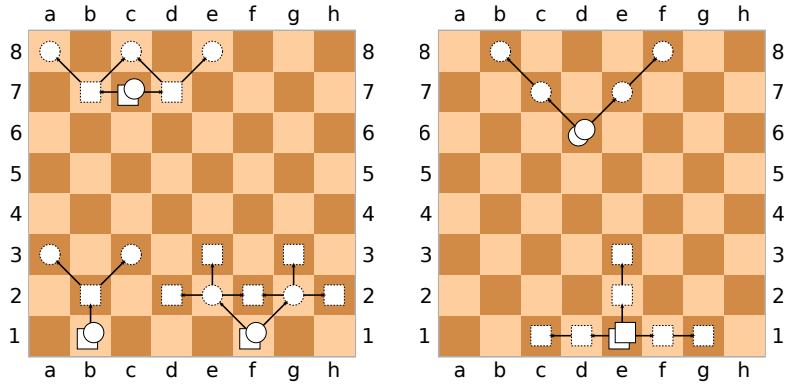


FIGURE 5 – Déploiement des pièces doubles

On ne peut pas capturer un pion ennemi pendant un déploiement. On ne peut donc pas déplier un pion sur une case occupée par un pion ennemi. Il est par contre possible de dépiler un pion sur une case occupée par un pion simple ou un double-pion ami : cela donne lieu à une nouvelle composition.

Rebonds

Quand un des bords latéraux de l'échiquier empêche un pion composé de poursuivre un mouvement ou un déploiement qu'il a entamé, il continue son mouvement en rebondissant contre ce bord. Les rebonds triviaux sont interdits, i.e. on ne peut pas commencer un mouvement ou un déploiement par un rebond.

Fin de partie

Le jeu se finit quand un ou plusieurs pions d'un joueur franchissent le bord opposé de l'échiquier, et le joueur en question a gagné. Cela peut se produire pendant un mouvement ou un déploiement.

On considère le jeu fini également quand un des joueurs finit par capturer tous les pions de l'autre : cas extrêmement rare où c'est le premier, bien entendu, qui est déclaré gagnant.

2 Interactivité et affichage

Toutes les options et tous les paramètres de votre programme doivent pouvoir être spécifiés en ligne de commande. Votre programme doit être muni d'un mode « jeu » et un mode « test » pour lancer des tests automatisés.

2.1 Mode jeu

Nous appelons « mode jeu » le mode normal du programme, c'est-à-dire où deux joueurs humains ou robots s'affrontent (soit 2 humains, soit 1 humain contre 1 robot, soit 2 robots).

En mode jeu, à chaque fois qu'un coup est joué, le nouvel état du plateau doit être affiché. Quand il est demandé à un joueur humain d'entrer un coup, faites bien attention à ce que l'état du plateau soit encore visible.

Il est demandé pour les joueurs humains d'accepter des entrées au format standard décrit plus bas, mais vous pouvez proposer d'autres façons supplémentaires d'entrer les coups si elles vous semblent plus intuitives.

Le mode de jeu par défaut est humain contre humain. L'option de ligne de commande pour affecter un type de joueur particulier au joueur blanc (resp. noir) est `-B <type>` (resp. `-N <type>`). Où le `<type>` est `humain` ou `robot` (mais vous pouvez en ajouter d'autres, en particulier si vous avez programmé plusieurs robots différents).

2.1.1 Représentation des coups

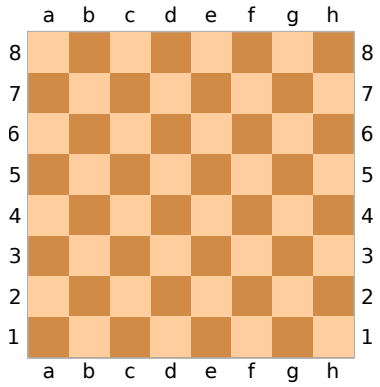


FIGURE 6 – Numérotation des cases

Comme aux échecs, chaque case de l'échiquier a un code correspondant à sa position, fait d'une lettre entre `a` et `h` dénotant sa colonne, suivie d'un chiffre entre `1` et `8` dénotant sa ligne.

Chaque coup sera noté sur une ligne propre, avec un retour à la ligne entre chaque coup. Les lignes paires correspondront donc aux coups du joueur blanc, et les impaires aux coups du joueur noir.

Mouvements On notera le déplacement d'une pièce simplement par sa position initiale, suivie de sa position finale, séparées par un tiret. Exemple : `b2-b3`. Si le coup fait sortir la pièce (et donc gagner le joueur), on se contentera de noter la case jouée, suivie d'un dièse, comme `g8#`.

Déploiements La notation du déploiement d'une pièce se décompose en trois parties :

— la position initiale de la pièce ;

— un caractère + ou * selon que le déploiement commence respectivement par les carrés ou par les ronds ;
— les positions des pièces déployées séparées par un tiret (autant de positions que la pièce composée possédait de sous-pièces).

Comme dans le cas du déplacement, un déploiement qui ferait sortir un pion se note simplement par la position du pion joué suivi d'un dièse.

On donne des exemples de déploiements à la figure 7.

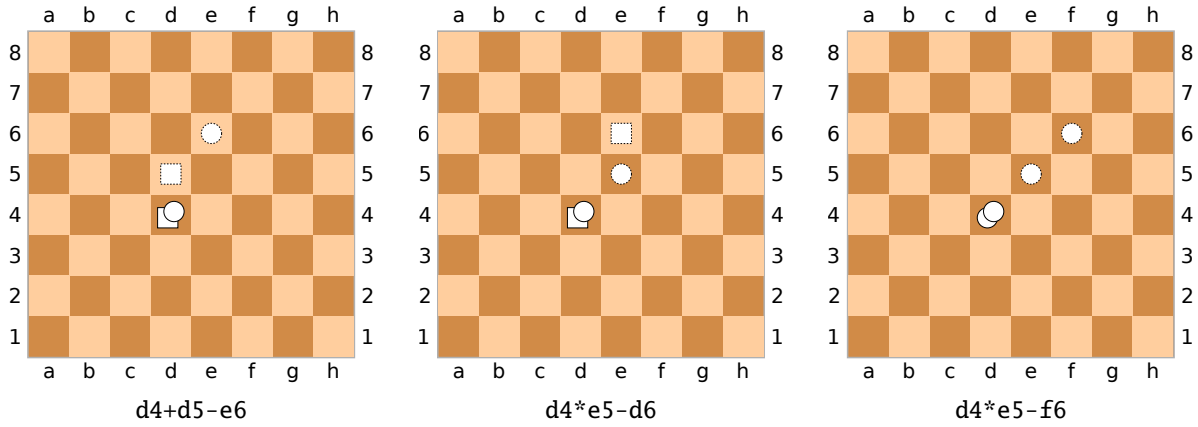


FIGURE 7 – Exemples de notations de déploiements à deux pièces

2.1.2 Affichage

Au début de la partie et après chaque coup joué, le plateau de jeu devra être affiché en texte sur la sortie standard. On devra pouvoir reconnaître l'échiquier et les pieces des deux couleurs, ainsi que les numéros des lignes et lettres des colonnes (système de coordonnées décrit plus haut).

2.1.3 Chargement d'une configuration

L'option `-c <fichier>` (facultative dans le projet) doit permettre de charger le jeu dans la configuration décrite dans le fichier `<fichier>`. Cette option peut servir à jouer sur une configuration initiale différente, ou tout simplement à restaurer une sauvegarde.

Pour faciliter l'échange de jeux de test, nous imposons un format pour le fichier de configuration. Il sera composé d'une suite de 64 pièces séparées par un retour à la ligne (caractère `'\n'`). L'ordre des pièces est donné en parcourant le tableau ligne par ligne, par numéros et lettres croissants, c'est-à-dire qu'on donnera les cases dans cet ordre : `a1`, `b1`, `c1`, `d1`, `e1`, `f1`, `g1`, `h1`, `a2`, `b2`, etc. Les pièces sont représentées par un code à deux caractères. Le premier caractère peut-être `B`, `N` ou `V`, selon respectivement que la pièce est blanche ou noire ou que la case est vide. Le deuxième caractère est un chiffre codant le type de la pièce, suivant le codage suivant.

Caractère	Pièce
`0`	case vide
`1`	carré simple
`2`	rond simple
`3`	carré double
`4`	rond double
`5`	carré triple
`6`	rond triple
`7`	carré-rond
`8`	carré-carré-rond
`9`	carré-rond-rond

On donne par exemple la configuration correspondant à la situation initiale dans le fichier `initial.conf`.

2.2 Mode test

Pour permettre de lancer des tests automatisés sur vos projets, nous vous demandons que votre programme soit muni d'un mode « test » (option `-t` en ligne de commande), de telle sorte que la commande

```
./gounki -t < test1.txt
```

(où `gounki` est le nom de votre programme, et `test1.txt` est un fichier texte contenant une liste de coups selon le format décrit ci-dessous) aura pour résultat de faire jouer à votre programme tous les coups du fichier `test1.txt`, dans l'ordre.

Nous imposons le format suivant : une liste de coups au format décrit plus haut, séparés des retours à la ligne. Par exemple :

```
a2-b3
g7-g6
b2-b3
h7-g6
b3+b4-c5
g6*f5-f4
```

Nous demandons en particulier que
— dans le mode test, aucune question ne soit posée à l'utilisateur, autre que les coups à jouer (l'entrée standard ne reçoit que les coups à jouer),
— après chaque coup, on affiche le plateau et le numéro du coup,
— quand le programme lit une entrée invalide ou un coup impossible, il quitte en retournant un code d'erreur.

3 Intelligence artificielle

Avoir un jeu fonctionnel, c'est bien, mais nous voulons permettre aux joueurs isolés de jouer sans devenir schizophrènes. Pour cela, vous allez implémenter une intelligence artificielle (IA) qui pourra jouer contre un joueur. Nous pourrons aussi faire jouer l'ordinateur contre lui-même, afin de tester nos différentes IA.

3.1 Robot aléatoire

Dans un premier temps, vous implémenterez l'une des IA la plus basique qui soit, à savoir celle quoi fait un choix aléatoire. Cette IA devra être capable de trouver tous les coups possibles, et d'en choisir un parmi ceux-ci.

Cette IA de base vous permettra de bien tester votre programme, et probablement de trouver un certain nombre de bugs. C'est aussi un bon échauffement pour la suite.

3.2 Heuristiques

On appelle configuration une disposition des pions sur le plateau. Ce jeu a un nombre fini de configurations, il serait donc possible en théorie de déterminer la meilleure stratégie à adopter pour chaque configuration. Cependant, calculer tous les coups pour connaître la stratégie gagnante représente un coût trop important (en temps et en mémoire). Une méthode commune consiste donc à définir une heuristique : une fonction qui associe un score à une configuration pour un joueur donné. Le joueur (le robot) cherchera à maximiser son heuristique.

Définition de l'heuristique de *votre* programme Il existe plusieurs heuristiques possibles, plus ou moins performantes. On notera C une configuration donnée, et η la fonction d'heuristique du joueur courant. Une heuristique doit respecter ces propriétés :

— Si la configuration est une défaite, $\eta(C) = 0$,
— Si la configuration est une victoire, $\eta(C) = \text{MAX}$[1].

On pourra enrichir l'heuristique avec ces différentes propositions :

— Valeur de l'armée : il est assez clair qu'avoir une pile de pions sous la main offre un certain nombre d'avantages par rapport à avoir plusieurs petits pions isolés. Les joueurs de Gounki semble s'accorder sur le fait que les meilleures combinaisons sont 3 pions mixtes (2 ronds + 1 carré, ou 2 carrés + 1 rond). On pourra affecter des coefficients aux piles de pions, et en déduire la valeur de l'armée.
— Percée : Le but est tout de même d'arriver au bout, on pourrait considérer l'avancée des forces d'attaque, leur nombre, et peut être la quantité de défenseurs qui se trouvent en face. Notez qu'il est parfois avantageux d'avancer plutôt que de tuer (une bonne percée vaut mieux qu'on grosse valeur de l'armée).
— Lignes de défense : un terrain clairsemé, ou trop déséquilibré peut être considéré comme désavantageux.
— Eléments isolés trop éloignés : un élément isolé en terrain ennemi (après la moitié du plateau) aura plus de mal à être remplacé (ou soutenu) qu'un élément proche du déploiement.

Vous êtes libres d'implémenter ou non ces propositions : c'est votre IA, faites qu'elle soit la meilleure possible, enrichissez la avec vos propres expériences, ajoutez vos propres règles, etc.

Utilisation dans l'IA Une fois l'heuristique définie, lorsque ce sera le tour de l'ordinateur, trouvez toutes les configurations dans laquelle vous pouvez vous trouver après une action, et appliquez l'heuristique à chacune de ces configurations. La configuration ayant l'heuristique maximum sera celle où vous choisirez d'aller.

Il peut être bon, si plusieurs actions possibles mènent à des configurations d'heuristiques proches, d'effecter un choix aléatoire parmi ces configurations, de sorte de choisir parmi les meilleures, plutôt que seulement la meilleure.

Test On pourra proposer, même en mode 2 joueurs, l'affichage de l'heuristique de la configuration courante pour chacun des joueurs. Les joueurs pourront alors voir leur avancement[2], et vous pourrez vous rendre compte des faiblesses de l'heuristique.

Dans un deuxième temps, lorsque vous aurez un robot capable d'utiliser une fonction d'heuristique, vous pourrez faire jouer l'ordinateur contre lui-même. Vous serez alors capable de confronter les différentes heuristiques. Avoir un peu d'aléatoire ici vous permettra de vraiment comparer les différentes heuristiques, de manière totalement automatique. N'hésitez pas à faire travailler votre ordinateur la nuit pour avoir les résultats de plusieurs simulations au petit matin.

1. MAX dépend de votre implémentation. C'est à vous de choisir le type de données représentant l'heuristique (entiers, flottants), la nombre de bits utilisés, etc.
2. Ou se moquer du programme !

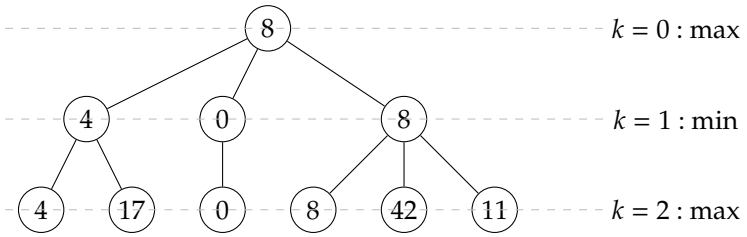
3.3 Stratégie min-max à N coups de profondeur

Dans un second temps, votre IA pourra prévoir plusieurs coups en avance. Si l'IA est capable de prévoir N coups d'avance, elle évaluera l'heuristique des configurations de niveau N (atteignables en N coups), puis calculera l'heuristique des configurations de niveau $k-1$ (le niveau 0 est la configuration courante du plateau) en prenant :

— le maximum des scores de niveau k atteignables depuis C si c'est au robot de jouer le niveau $k-1$,

— le minimum des scores de niveau k si c'est à l'adversaire de jouer $k-1$.

Par exemple, si $N = 2$:



3.4 Performances, extensions et autre

Toute extension est bien sûr la bienvenue : propositions de meilleures stratégies, sélection du niveau de difficulté, amélioration des performances ou autre. Il en sera bien sûr tenu compte dans la note.

Par exemple, lorsque le robot choisit une action, il va être amené à recalculer les différentes configurations à partir de sa position actuelle. Si jamais $N > 2$, les configurations de 0 à $N-2$ ont déjà été calculées. On pourra essayer de s'en souvenir, pour ne pas les recalculer.

3.5 Généralités d'évaluation

Il sera tenu compte de la modularité de votre programme (pouvoir utiliser facilement une autre heuristique), de ses fonctionalités, de la qualité de l'heuristique, de vos tests (on aime les jolis graphes, et tant qu'à avoir fait le travail de coder différentes heuristiques et d'avoir choisi la meilleure, pourquoi ne pas faire un graphe récapitulatif dans le rapport), etc.

4 Consignes

4.1 Fonctionnalités demandées

Obligatoires :

— mode jeu acceptant permettant de faire affronter toutes les combinaisons de joeur(s) humain(s) et de robot(s) aléatoire(s),

— mode test,

— dans les deux cas, règles standard avec position de départ classique.

En particulier, vous porterez le plus grand soin à

— implémenter les règles du jeu avec précision (ne pas accepter les coups illégaux, calculer le bon résultat pour les coups légaux, détecter la fin de partie),

— afficher un échiquier clair, avec coordonnées visibles,

— respecter le format d'entrée proposé (sans quoi il sera difficile de tester votre programme).

Optionnelles :

- chargement de la position initiale depuis un fichier,
- intelligence artificielle (avec les différentes heuristiques imaginables), choix entre IA et 2 joueurs par option de ligne de commande,
- fonction « annuler » (pour revenir un ou plusieurs coups en arrière),
- fonction « rejouer » (contraire d'« annuler »),
- fonction sauvegarde (soit juste le dernier état du plateau, soit tout l'historique des coups) et fonction correspondante de restauration de sauvegarde,
- toute autre fonctionnalité qui vous semblerait utile dans un tel jeu.

4.2 Consignes générales

- Le projet est un travail original à réaliser seul ou en binôme.
- Le projet doit être un programme en langage C. Il doit respecter la norme C89. En particulier il doit pouvoir être compilé sans avertissement ni erreur avec GCC, en passant les options `-std=c89 -Wall -pedantic`.
- Il est recommandé d'organiser le projet en plusieurs modules avec compilation séparée.
- Vous devez écrire un fichier `Makefile` permettant de compiler votre projet par un simple appel de la commande `make`.
- Le rendu se fait sur DidEL, dans la section « Travaux ». Vous devrez y déposer votre projet sous forme d'une archive zip contenant au moins
  - votre code source (fichiers `.c` et `.h`),
  - votre fichier `Makefile`,
  - un fichier `README` contentant toute explication que vous jugerez nécessaire à la compilation et l'utilisation du projet par d'autres personnes que vous,
  - et un bref rapport (4 pages maximum, format PDF), qui doit lister les fonctionnalités que vous avez programmées, en expliquant à chaque fois comment vous avez interprété la consigne.

Le jury appréciera la qualité du code source de votre projet notamment sur les critères suivants : clarté, concision, organisation logique en fonctions et fichiers, efficacité et correction des algorithmes et bonne gestion de la mémoire.