Algorithme min-max et accélération alpha-beta

Lancelot Pecquet

25 novembre 2004

1 Modélisation des jeux de réflexion

On considère un jeu de réflexion ou deux joueurs A et B jouent alternativement. L'**ensemble des états possibles du jeu** est noté E. À chaque configuration du jeu e est associée la valeur $J(e)$ qui vaut A si le coup précédent a été joué par B et B si le coup précédent a été joué par A, en convenant que $J(e_0) = A$ où e_0 est l'état initial de la partie. On notera également $S(e)$ l'ensemble des configurations qui jouables à partir de e à ce tour. Une configuration e est dite **finale** si $S(e) = \varnothing$. On notera F l'ensemble de ces configurations finales.

En début de partie, le joueur A choisit une configuration e_1 dans $S(e_0)$, puis c'est au tour du joueur B de choisir une configuration e_2 dans $S(e_1)$, et ainsi de suite. La partie se termine lorsque le coup de l'un des joueurs conduit à une configuration finale. Une fois la partie terminée dans l'état $f \in F$, le joueur A a un **score** $s_A(f)$ et le joueur B a un score $s_B(f)$. Les scores sont des réels positifs ou nuls, ou bien $+\infty$. Le gagnant est celui des deux joueurs qui a le plus haut score. Si les scores sont égaux, les joueurs sont *ex æquo*.

À *Puissance 4*, on peut définir les scores de fin de partie de plusieurs manières, par exemple :

- **ex1 :** $s_A(e) = +\infty$ (resp. $s_B(e) = +\infty$) si la configuration e présente quatre ronds alignés (respectivement quatre croix alignées) et $s_A(e) = 0$, (resp. $s_B(e) = 0$) sinon ;
- **ex2 :** $s_A(e)$ (resp. $s_B(e)$) est le maximum du nombre de ronds alignés présents (resp. croix alignées présentes) dans la configuration finale e.
- **ex3 :** on définit :
  1. $s_A(e) = 1$ (resp. $s_B(e) = 1$) si la configuration ne fait apparaître de des ronds isolés (resp. croix isolées) ;
  2. $s_A(e) = 2$ (resp. $s_B(e) = 2$) si la configuration fait apparaître, au mieux, deux ronds contigus, (resp. deux croix contiguës) ;
  3. $s_A(e) = 3$ (resp. $s_B(e) = 3$) si la configuration fait apparaître, au mieux, trois ronds alignés, (resp. trois croix alignés) ;
  4. $s_A(e) = +\infty$ (resp. $s_B(e) = +\infty$) si la configuration fait apparaître quatre ronds alignés, (resp. quatre croix alignés).

ou encore d'autres manières.

2 Algorithme min-max

2.1 Fonction d'évaluation

L'algorithme min-max permet à un ordinateur de jouer de manière performante à ce type de jeu, en anticipant sur ce que sera la situation plusieurs coups à l'avance. L'idée initiale consiste à définir une **fonction d'évaluation** $v : E \longrightarrow \mathbb{R}_+ \sqcup \{+\infty\}$ telle que :

1. pour toute position finale $f \in F$, on ait $v(f) = s_A(f)/s_B(f)$. L'objectif du joueur A revient donc à maximiser $v(f)$, celui de B est de minimiser $v(f)$. Si $v(f) > 1$, le joueur A a gagné, si $v(f) < 1$, c'est le joueur B qui a gagné. Si $v(f) = 1$, les joueurs sont *ex æquo* (on prendra donc la convention que $0/0 = 1$ et que $\infty/\infty = 1$ dans la définition de v).

2. pour toute position $e \in E$, la valeur $v(e)$ soit d'autant plus grande que la configuration est « favorable » à A et d'autant plus petite que la configuration est « favorable » à B.

Le fait qu'un état e soit « favorable » signifie que le joueur $J(e)$ peut raisonnablement espérer pouvoir aboutir à une position finale gagnante à partir de e. Reste encore à définir plus précisément ce qu'on entend par là.

2.2 Hypothèse de travail et définition de la fonction d'évaluation

Il est possible qu'une position soit favorable, sans l'ombre d'un doute, si, quoi que fasse l'adversaire, on finira par gagner. En revanche, dans la plupart des cas, le fait qu'une position soit favorable ou pas dépend énormément de la manière dont l'adversaire va se comporter. En définissant la fonction v par le fait que, pour tout e :

$$v(e) = \begin{cases} s_A(e)/s_B(e) & \text{si } e \in F\ ; \\ \alpha(e) & \text{si } J(e) = A \text{ avec } \alpha(e) = \max\limits_{e' \in S(e)} v(e')\ ; \\ \beta(e) & \text{si } J(e) = B \text{ avec } \beta(e) = \min\limits_{e' \in S(e)} v(e')\ ; \end{cases}$$

une façon simple de modéliser le point 2. de la Section 2.1 consiste à supposer que les adversaires se comportent de la manière suivante. À partir d'une position e :

- A choisit toujours un état $e' \in S(e)$ qui atteint le maximum $\alpha(e)$ lorsque c'est à lui de jouer ;
- B choisit toujours un état $e' \in S(e)$ qui atteint le minimum $\beta(e)$ lorsque c'est à lui de jouer.

2.3 Comment joue-t-on ?

Si l'on dispose d'une fonction v, définie comme précédemment, une machine implémentant l'algorithme min-max joue selon l'Algorithme 1.

Algorithme 1 Algorithme permettant de choisir un état de jeu en exploitant une fonction d'évaluation $v(e)$.

fonction choix-depuis(e)
  si $J(e) = A$ **alors**
    `résultat` $\leftarrow e'$ parmi les $S(e)$ tel que $v(e')$ est maximal ;
  sinon
    `résultat` $\leftarrow e'$ parmi les $S(e)$ tel que $v(e')$ est minimal ;
  fin si
  retourner `résultat` ;
fin fonction

Dans le cas où plusieurs états conduisent au maximum (respectivement, au minimum), la machine pourra choisir aléatoirement parmi ceux-là. Si la machine joue contre un humain, celle-ci n'aura qu'à faire ses choix par rapport à l'état dans lequel l'humain l'aura placé au coup précédent.

2.4 Algorithme de calcul de la fonction d'évaluation v

Bien que l'arbre suivant ne doive pas être construit explicitement, il peut être utile de le définir pour comprendre la situation. Considérons donc T, l'arbre des configurations du jeu, c'est-à-dire l'arbre dont les nœuds sont étiquetés par E avec les propriétés que :

1. la racine de T est étiquetée par e_0 ;

2. étant donné un nœud étiqueté par e, ses fils sont étiquetés par les éléments de $S(e)$ correspondant aux configurations accessibles en jouant un coup à partir de e.

Le principe consiste à étiqueter chaque feuille correspondant à un état e' dans l'arbre T par la valeur $v(e')$, puis d'étiqueter le nœud père, correspondant à un état e par :

– le maximum des $v(e')$ où e' décrit les états qui étiquètent les fils de e, si $J(e) = A$;

– le minimum des $v(e')$ où e' décrit les états qui étiquètent les fils de e, si $J(e) = B$.

et de recommencer, ainsi de suite, jusqu'à atteindre la racine de l'arbre, ainsi que cela est indiqué dans la Fig. 1.

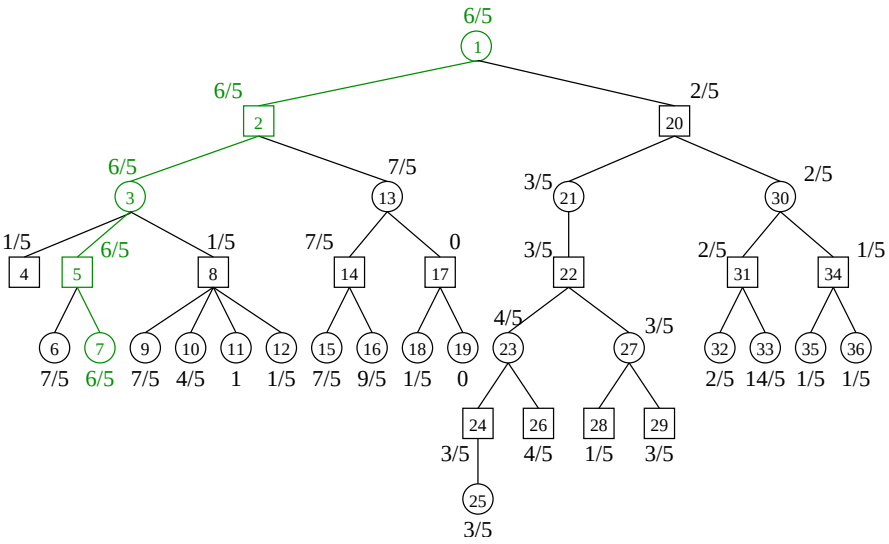

FIG. 1 – On écrit $v(e) = s_A(e)/s_B(e)$ pour chaque état final e correspondant à une feuille de l'arbre. On propage ensuite alternativement le maximum et le minimum de ces valeurs en remontant le long de l'arbre jusqu'à la racine. Les sommets sont numérotés dans l'ordre du parcours en pronfondeur de l'arbre qui sera fait par l'Algorithme 2. Une fois faits tous les calculs, la stratégie de jeu consiste, pour A, à choisir toujours le coup e' qui maximise $v(e')$ et, pour B, à choisir toujours le coup e' qui minimise $v(e')$. Si le joueur A joue contre la machine B, il peut gagner en choisissant l'état 2 qui maximise v, puis la machine B choisirai l'état 3 qui minimise v, ensuite A joue dans l'état 5 pour maximiser v, puis la machine B choisira l'état 7 pour minimiser v. Au final, A gagnera sur B avec un rapport des scores $s_A(e)/s_B(e) = 6/5$ (parcours indiqué en vert).

Algorithme 2 Algorithme min-max pour calculer v, version brute

```
fonction eval(e)
  si e ∈ F alors
    // Dans un état final, on calcule le quotient des scores :
    resultat ← s_A(e)/s_B(e) ;
  sinon
    si J(e) = A alors
        // On calcule α, le max des v(e') parmi les fils e' de e :
        α ← 0 ;
        pour e' ∈ S(e) faire
          α ← max(α, eval(e')) ;
        fin pour ;
        resultat ← α ;
    sinon
        // On calcule β, le min des v(e') parmi les fils e' de e :
        β ← +∞ ;
        pour e' ∈ S(e) faire
          β ← min(β, eval(e')) ;
        fin pour ;
        resultat ← β ;
    fin si ;
  fin si ;
  retourner resultat ;
fin fonction ;
```

2.5 Calcul à profondeur fixée et heuristique d'évaluation

La complexité de l'algorithme min-max, dans sa version brute, est évidemment exponentielle en le nombre de coups qu'il faut jouer pour terminer la partie. Aussi, en pratique, il n'est souvent pas possible d'aller explorer toutes les possibilités qui sont en nombre trop grand. On arrête donc le calcul à une profondeur fixée p, permettant que ce calcul se fasse en temps raisonnable.

Le problème, c'est qu'*a priori*, la notion de score n'existe que pour les états finaux et qu'on ne peut donc pas appliquer le calcul tel que précédemment.

Il va donc falloir définir des *scores temporaires* $s'_A(e)$ et $s'_B(e)$ qui permettront de dire, à tout moment, si l'état e est plutôt favorable à A par rapport à B, ou pas. Ces scores temporaires devront coïncider avec les scores réels dans le cas où un état est final. Dans certains jeux, la notion de score temporaire est très claire, dans d'autre moins (échecs !) et il faut trouver expérimentalement une mesure satisfaisante.

À *Puissance 4*, on peut définir les scores temporaires de plusieurs manières, par exemple en prolongeant les fonctions scores définies dans les exemples précédents pour les fin de partie à toute configuration intermédiaire car c'est possible ici :

- **ex1 :** $s'_A(e) = +\infty$ (resp. $s'_B(e) = +\infty$) si la configuration e présente quatre ronds alignés (respectivement quatre croix alignées) et $s'_A(e) = 0$, (resp. $s'_B(e) = 0$) sinon ;
- **ex2 :** $s'_A(e)$ (resp. $s'_B(e)$) est le maximum du nombre de ronds alignés présents (resp. croix alignées présentes) dans la configuration finale e.
- **ex3 :** on définit :
  1. $s'_A(e) = 1$ (resp. $s'_B(e) = 1$) si la configuration ne fait apparaître de des ronds isolés (resp. croix isolées) ;
  2. $s'_A(e) = 2$ (resp. $s'_B(e) = 2$) si la configuration fait apparaître, au mieux, deux ronds contigus, (resp. deux croix contiguës) ;
  3. $s'_A(e) = 3$ (resp. $s'_B(e) = 3$) si la configuration fait apparaître, au mieux, trois ronds alignés, (resp. trois croix alignés) ;
  4. $s'_A(e) = +\infty$ (resp. $s'_B(e) = +\infty$) si la configuration fait apparaître quatre ronds alignés, (resp. quatre croix alignés).

ou encore d'autres manières.

On définit ensuite une fonction w, dite *heuristique* (c'est-à-dire une fonction dont le comportement sera raisonnablement proche de ce qu'on veut mais qu'on ne peut pas atteindre exactement), valant $s'_A(e)/s'_B(e)$ pour les états e qui sont à profondeur $p_{\max}$ ou bien qui sont finaux, et qu'on définit de manière analogue à v. Ainsi, de la même façon, pour toute position $e \in E$, la valeur $w(e)$ soit d'autant plus grande que la configuration est « favorable » à A et d'autant plus petite que la configuration est « favorable » à B. On définit donc :

$$w(e) = \begin{cases} s'_A(e)/s'_B(e) & \text{si } e \in F \text{ ou si } e \text{ est à profondeur } p_{\max} ; \\ \alpha(e) & \text{si } J(e) = A \text{ avec } \alpha(e) = \max\limits_{e' \in S(e)} w(e') ; \\ \beta(e) & \text{si } J(e) = B \text{ avec } \beta(e) = \min\limits_{e' \in S(e)} w(e') ; \end{cases}$$

que l'on calcule grâce à l'Algorithme 3 qui utilise lui-même la fonction auxilliaire calculée par l'Algorithme 4.

Algorithme 3 Algorithme min-max à profondeur fixée, fonction principale

fonction eval(e**,** $p_{\max}$**)**
  retourner eval-aux$(e, 0)$;
fin fonction ;

Algorithme 4 Algorithme min-max à profondeur fixée, fonction auxiliaire.

fonction eval-aux(e, i)
  si $e \in F$ **ou si** $i = p_{\max}$ **alors**
    // Dans un état final, on calcule le quotient des scores :
    `resultat` $\leftarrow w(e)$;
  sinon
    si $J(e) = A$ **alors**
      // On calcule α, le max des $v(e')$ parmi les fils e' de e :
      $\alpha \leftarrow 0$;
      pour $e' \in S(e)$ **faire**
        $\alpha \leftarrow \max\big(\alpha, \textbf{eval-aux}(e', i+1)\big)$;
      fin pour ;
      `resultat` $\leftarrow \alpha$;
    sinon
      // On calcule β, le min des $v(e')$ parmi les fils e' de e :
      $\beta \leftarrow +\infty$;
      pour $e' \in S(e)$ **faire**
        $\beta \leftarrow \min\big(\beta, \textbf{eval-aux}(e', i+1)\big)$;
      fin pour ;
      `resultat` $\leftarrow \beta$;
    fin si ;
  fin si ;
  retourner `resultat` ;
fin fonction ;

2.6 Accélération alpha-beta en bornant la fonction d'évaluation

En fait, lors du calcul de v ou de w, on fait des calculs inutiles. En effet, on peut souvent savoir qu'on a déjà trouvé un maximum (*cf.* Fig. 2), ou un minimum (*cf.* Fig. 3) sans avoir besoin d'explorer tout l'arbre des possibilités.

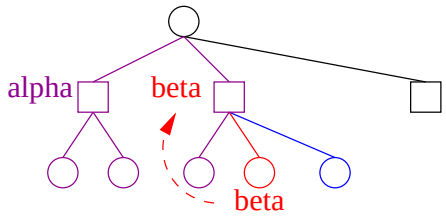

FIG. 2 – Alors qu'on est en train de calculer un maximum α, au niveau intermédiaire, le minimum β qui est en train d'être calculé au niveau inférieur prend une valeur inférieure ou égale à α. Dès lors, il n'est plus nécessaire de continuer car toutes les autres valeurs de β seront inférieures ou égales à la valeur actuelle et la valeur de α sera donc inchangée. On remonte donc β au niveau intermédiaire et on élague les branches qui n'ont plus à être explorées (en bleu).

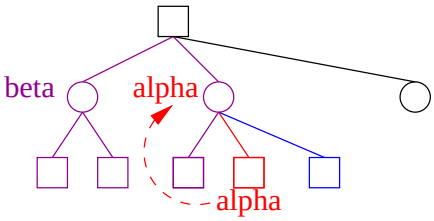

FIG. 3 – Alors qu'on est en train de calculer un minimum β, au niveau intermédiaire, le maximum α qui est en train d'être calculé au niveau inférieur prend une valeur supérieure ou égale à β. Dès lors, il n'est plus nécessaire de continuer car toutes les autres valeurs de α seront supérieures ou égales à la valeur actuelle et la valeur de β sera donc inchangée. On remonte donc α au niveau intermédiaire et on élague les branches qui n'ont plus à être explorées (en bleu).

Le résultat de cet élagage appliqué à l'exemple présenté est défini dans la Fig 4.

On formalise cette astuce avec l'Algorithme 5, lequel utilise une fonction auxiliaire calculée dans l'Algorithme 6. Une version à profondeur fixée peut également être définie.

Algorithme 5 Algorithme alpha-beta pour calculer v, fonction principale. Seules les valeurs « inintéressantes » ne sont pas calculées.

fonction eval-approx(e)
  retourner eval-approx-aux(e, 0, $+\infty$) ;
fin fonction ;

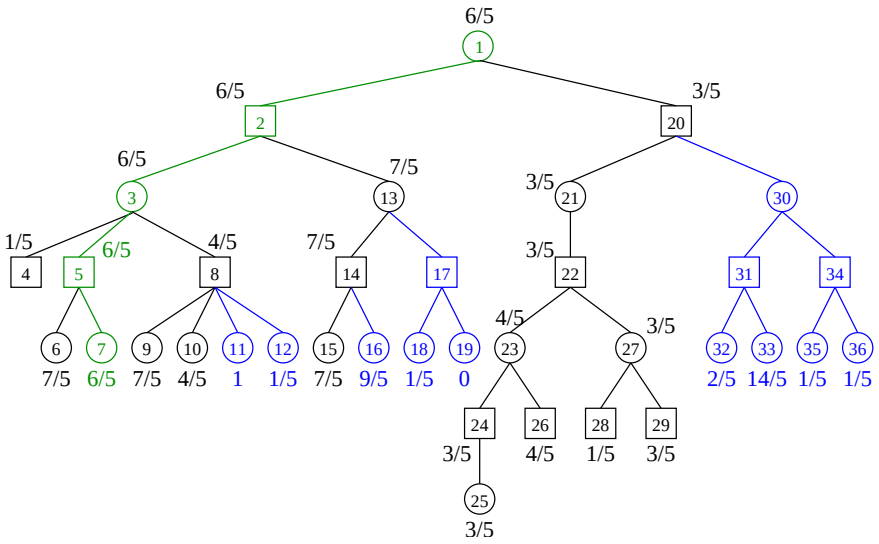

FIG. 4 – En appliquant les élagages alpha-beta la Fig. 1 devient celle-ci. De même, une fois faits tous les calculs, la stratégie de jeu consiste, pour A, à choisir toujours le coup e' qui maximise $v(e')$ et, pour B, à choisir toujours le coup e' qui minimise $v(e')$. Si le joueur A joue contre la machine B, il peut gagner en choisissant l'état 2 qui maximise v, puis la machine B choisirai l'état 3 qui minimise v, ensuite A joue dans l'état 5 pour maximiser v, puis la machine B choisira l'état 7 pour minimiser v. Au final, A gagnera sur B avec un rapport des scores $s_A(e)/s_B(e) = 6/5$ (parcours indiqué en vert) : la stratégie est bien la même que dans la Fig. 1 bien que, par exemple, dans la case 20 soit ici étiquetée par la valeur 3/5 alors qu'elle l'était par 2/5 dans la Fig. 1 : peu importe car le maximum est toujours 6/5. Un phénomène analogue a lieu dans la case 8.

Algorithme 6 Algorithme alpha-beta pour calculer v, fonction auxilliaire.

```
fonction eval-approx-aux(e, α, β)
  si e ∈ F alors
    // Dans un état final, on calcule le quotient des scores :
    resultat ← s_A(e)/s_B(e) ;
  sinon
    si J(e) = A alors
        // On calcule α, le max des v(e') parmi les fils e' de e :
        pour e' ∈ S(e) faire
          si α < β alors
              α ← max(α, eval-approx-aux(e', α, β)) ;
          sinon // on élague comme dans la Fig. 3
              sortir de la boucle ;
          fin si ;
        fin pour ;
        resultat ← α ;
    sinon
        // On calcule β, le min des v(e') parmi les fils e' de e :
        pour e' ∈ S(e) faire
          si α < β alors
              β ← min(β, eval-approx-aux(e', α, β)) ;
          sinon // on élague comme dans la Fig. 2
              sortir de la boucle ;
          fin si ;
        fin pour ;
    resultat ← β ;
  fin si ;
  retourner resultat ;
fin fonction ;
```

2.7 Optimisation de la méthode alpha-beta

Il est clair que si les éléments de $S(e)$ sont triés par ordre croissant de $v(e)$ pour $J(e) = A$, le max et le min se calculent en temps constant et on pourrait aussi savoir, grâce à une recherche dichotomique, en temps logarithmique en $|S(e)|$, les éléments à partir desquels la recherche doit se poursuivre, et ceux à éliminer.

En pratique, on ne peut pas forcément attendre d'avoir tous les nœuds terminaux pour pouvoir les trier puisque, précisément, on essaie de ne pas tous les calculer. Dans une version plus sophistiquée, on pourrait utiliser des calculs précédents et l'heuristique pour faire un pré-tri.

Une version à profondeur fixée se déduit de manière analogue à celle définie précédemment. Une bonne utilisation d'alpha-beta permet de doubler la profondeur de recherche en général.

3 Critères d'évaluation du projet

3.1 Calcul de la note globale

Chaque binôme aura 10 min pour présenter son travail et 5 min de questions. La note du projet sera calculée de la manière suivante :

$$\text{note} = \frac{\texttt{round}(\text{note du rapport} + \text{note du mode d'emploi} + 2 \cdot \text{note du programme} + 3 \cdot \text{note d'oral})}{7}$$

où
- `round` est la fonction qui arrondit à l'entier le plus proche, c'est-à-dire que $\texttt{round}(x)$ est l'entier n tel que $|x - n|$ est minimal si $x \neq k/2$ pour $k \in \mathbb{N}$ et qui vaut $n = \lceil k/2 \rceil$ sinon ;
- le coefficient 3 à l'oral permet de discriminer, au sein d'un binôme, ceux qui ont plus ou moins travaillé et/ou compris ;
- le coefficient 2 au programme permet de donner une importance plus grande à celui-ci par rapport au mode d'emploi et au rapport (qui sont tout de même importants !) ;

3.2 Calcul de la note du rapport

Le rapport a pour objectif de présenter l'ensemble du travail réalisé. Il comportera une douzaine de pages maximum (hors annexe) et sera structuré de la manière suivante :

1. **un résumé** d'une demi-page, reprenant la problématique générale et vos résultats : celui-ci permet au lecteur de comprendre de quoi il s'agit en moins d'une minute ;
2. **une table des matières** ;
3. **une introduction** d'une ou deux pages, présentant brièvement le contexte (jeu de réflexion, règles,...), la problématique (différents points du cahier des charges), le travail réalisé et le plan du rapport ;
4. **le corpus** décrivant en détail, en une dizaine de pages maximum, votre travail. Vous prendrez soin, en particulier, de préciser :
   (a) la description théorique des structures de données et des algorithmes retenus ;
   (b) l'organisation pratique du programme : structures de données, procédures et fonctions,
   (c) les résultats d'expériences, permettant de montrer les limites de votre travail.
5. **une conclusion** d'une page dans laquelle, après avoir brièvement rappelé l'objectif à atteindre, vous présenterez les résultats obtenus et les perspectives sur une suite éventuelle de ces travaux ;
6. **un index**, éventuellement ;
7. **une bibliographie**, éventuellement ;
8. **des annexes**, comprenant, en particulier
   (a) **les listings complets** ;
   (b) **le mode d'emploi** (noté à part, voir section suivante).

Les critères d'évaluation sont :

1. **le respect de la structure demandée** : résumé, introduction,...
2. **la précision des résultats :** on sait exactement à quoi on a vraiment abouti et où sont les limites du travail réalisé ;
3. **la clarté de la démarche** : les explications sur les aspects théoriques et pratiques sont claires ;

4. **la clarté d'expression** : le texte suit un cheminement logique, la grammaire et l'orthographe sont respectées ;
5. **la qualité de la réalisation** : mise en page, illustrations par des dessins, photos, captures d'écrans,...

En gros, on peut dire que :

1. un rapport qui ne respecte pas la structure demandée a moins de 08/20 ;
2. un rapport qui ne décrit pas bien ce qui a vraiment été fait a moins de 10/20 ;
3. un rapport qui satisfait les trois premiers points précédents peut avoir une note entre 10 et 20 selon le respect des critères précédents.

3.3 Calcul de la note du mode d'emploi

Le mode d'emploi doit faire de une à deux pages. Il a pour objectif de permettre à une personne de tester le logiciel en une ou deux minutes. Il présentera :

1. **un résumé** des règles du jeu ;
2. **la liste des fichiers** nécessaires à la compilation et à l'exécution du logiciel et la manière de compiler les sources ;
3. **la syntaxe requise pour lancer le programme**, avec ses éventuelles options ;
4. **la syntaxe requise à l'intérieur** du programme pour utiliser celui-ci ;
5. **la syntaxe de sortie du programme**.

Les critères d'évaluation sont :

1. **le respect de la structure demandée** pour le document ;
2. **l'aspect synthétique** du document ;
3. **la clarté d'expression** : le texte suit un cheminement logique, la grammaire et l'orthographe sont respectées ;
4. **la qualité de la réalisation** : mise en page, illustrations par des dessins, photos, captures d'écrans,...

En gros, on peut dire que :

1. un mode d'emploi qui ne respecte pas la structure demandée a moins de 08/20 ;
2. un mode d'emploi qui ne permet pas d'utiliser le programme en quelques minutes sans avoir besoin de regarder ailleurs a moins de 10/20 ;
3. un mode d'emploi qui satisfait ces critères peut avoir une note entre 10 et 20 selon le degré de respect des critères précédents.

3.4 Calcul de la note du programme

La note de votre programme consistera en une mesure de l'ensemble des critères suivants :

1. **conformité avec le cahier des charges :** chacune des fonctionnalités demandées est réalisée ;
2. **structures de données :** celles-ci doivent être pertinentes et claires ;
3. **algorithmique :** les méthodes répondent au problème posé de manière efficace ;
4. **robustesse :** le programme ne plante pas, même si on ne rentre pas les bonnes valeurs (l'utilisateur est invité à modifier ses choix s'il s'est trompé) ;

5. **modularité :** le problème est bien découpé en fonctions et procédures (voire packages) pertinentes pour l'initialisation, l'affichage, les différentes tâches à réaliser ;
6. **clarté :** le code source est lisible et judicieusement commenté, les variables ont des noms pertinents ;
7. **adaptabilité :** le code est facilement adaptable à des situations voisines (pas de constantes numériques) ;
8. **ergonomie :** le logiciel est d'un maniement aisé et naturel.

En gros, on peut dire que :

1. un programme qui ne compile pas a moins de 08/20 ;
2. un programme qui ne satisfait pas le cahier des charges a moins de 10/20 ;
3. un programme qui satisfait le cahier des charges peut avoir entre 10 et 20 selon le degré de respect des critères cités et de l'efficacité des méthodes retenues.

3.5 Calcul de la note d'oral

L'oral consistera en une démonstration du logiciel. On notera :

1. **l'aisance** dans la manipulation du programme ;
2. **la compréhension** du fonctionnement interne du programme ;
3. **la mise en évidence des qualités du programme**, dans l'ordre prévu à la section précédente (respect du cahier des charges,. . .,adaptabilité) ;
4. **le respect du timing** (5min par candidat) ;
5. **la maîtrise des aspects théoriques** du sujet, en particulier à la lecture du rapport ;
6. **la clarté de l'expression**.

En gros, on peut dire que, sous réserve que le programme satisfasse le cahier des charges :

1. un étudiant qui ne sait pas manipuler son programme a moins de 08/20 ;
2. un étudiant qui ne comprend pas l'essentiel de son programme a moins de 10/20 ;
3. un étudiant qui sait manipuler son programme, qui comprend la manière dont il fonctionne du point de vue de la programmation et du point de vue théorique et qui réussit clairement à mettre en évidence les qualités du programme dans le temps imparti a une note qui peut tendre vers 20/20.