M1 Informatique - Module Algorithmique et complexité

Rapport

Intelligence Artificielle pour le Mastermind

Benoît MARECHAL et Marc RASTOIX

Encadrant : Jean-Jacques CHABRIER

Table des matières

Table des figures

1 Introduction

1.1 Rappel du sujet

Dans le cadre du module d'Algorithmique et complexité du Master 1 d'Informatique, nous avons à réaliser le projet d'intelligence artificielle pour le Mastermind. Le but de ce projet et de proposer et de programmer les meilleures stratégies possibles de résolution de ce jeu. Le programme doit donc tirer au hasard une combinaison puis la deviner en posant le minimum de questions possibles. On cherche à connaître ce nombre de questions ainsi que l'efficacité de nos algorithmes à travers leur complexité, leur temps d'exécution et leur nombre de tentatives maximales.

1.2 Historique et règles du mastermind

Le Mastermind est un jeu de société inventé par Marco Meirovitz dans les années 60. C'est un jeu de réflexion et de déduction à deux joueurs. Au même titre que les échecs, Mastermind fait appel à la logique et à la mémoire.

Le principe du jeu est relativement simple. Le joueur 1 crée une combinaison de K pions avec N couleurs. Le joueur 2 doit trouver la combinaison secrète. Pour cela il dispose d'un nombre limité de tentatives. A chaque test le joueur 1 indique le nombre de pions bien placés et le nombre de pions mal placés (c'est-à-dire les poins dont la couleur est présente dans la combinaison, mais dont l'emplacement choisi et mauvais). Si le joueur 2 trouve la combinaison secrète il gagne la manche. Ensuite les deux joueurs intervertissent les rôles.

Voici un exemple de partie :

On a cet ensemble de couleurs à disposition :

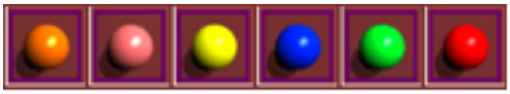

FIG. 1 – Jeux de couleurs, ici on a 6 couleurs donc N = 6

Ceci est la combinaison à deviner :

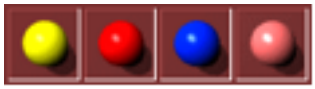

FIG. 2 – Exemple de combinaison, ici il y a 4 cases donc K = 4

Dans l'exemple de partie suivant, le pion noir indique le nombre de pions bien placés et le pion blanc indique le nombre de pions mal placés dans la combinaison proposée :

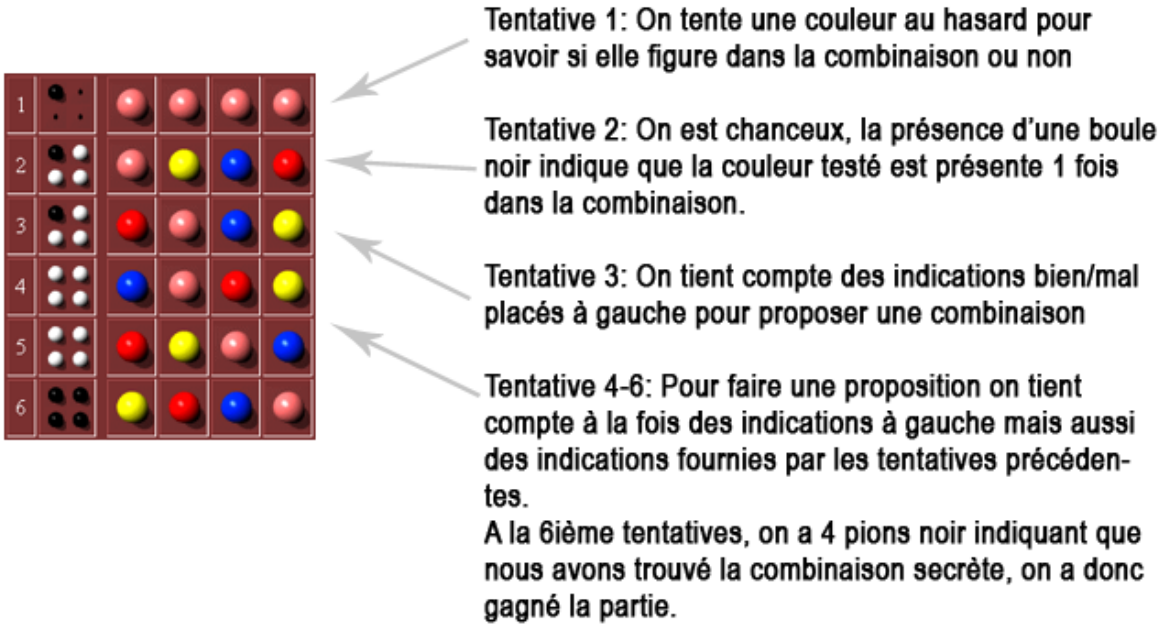

FIG. 3 – Exemple de résolution de mastermind

1.3 Objectifs du projet

1.3.1 Objectifs de base

- Réalisations des meilleures stratégies de résolutions du Mastermind.
- Répondre à la question : "Quel est le nombre minimum de questions à poser dans le pire des cas (4 ou 5 semble-t-il ?)"

1.3.2 Objectifs supplémentaires que nous nous sommes fixé

- Réalisation d'une étude mathématique sur la théorie du Mastermind.
- Rapport rédigé en LaTeX
- Pas de source, nous avons utilisé le pseudo code, pour décrire les algorithmes.
- Utilisation de "vrai" langage de programmation, nous avons retenu le language Java.
- Rapport compréhensible par un autre étudiant, qui doit être capable, après lecture, d'écrire l'algorithme de résolution.
- Création d'un interface graphique de mise en concurrence des stratégies
- Réalisation de jeux de tests et statistiques entre les stratégies

2 Etude mathématique sur le Mastermind

Avant d'expliquer les algorithmes mis en œuvre pour résoudre une combinaison. Nous allons effectuer ici une étude mathématique sur le Mastermind, afin de répondre à la question : "Quel est le nombre minimum de questions à poser dans le pire des cas (4 ou 5 semble-t-il ?)"

Pour répondre à cette question, nous nous sommes aidé de la formule donnée en Travaux dirigé, dans l'exercice sur les pièces. A savoir, que le nombre maximal de tentatives effectuées pour résoudre un problème est égal au log_2 du nombre de combinaisons possibles, divisé par le log_2 du nombre d'indications fournis à chaque tentative.

Nous allons donc déterminer le nombre de combinaison et d'indications possibles, en fonction des règles du Mastermind. Les formules seront données dans le cas général, afin de pouvoir les appliquer à tout type de Mastermind.

2.1 Recherche du nombre de combinaisons possible

En combinatoire, lorsque l'ordre d'apparition de N objets distincts répartis sur K emplacements est pris en compte, le nombre de combinaisons possible s'appelle un Arrangement.

Au Mastermind, les N objets distincts correspondent aux boules de couleurs, et les K emplacements correspondent aux cases dans lesquelles on place les boules de couleurs.
Aussi lorsque les répétitions d'objets distincts sont autorisées nous avons :

$$nombre\_de\_combinaisons = n^k$$

Et lorsque les répétitions d'objets distincts sont interdites nous avons :

$$nombre\_de\_combinaisons = A_n^k = \frac{n!}{(n-k)!}$$

Sachant que selon les règles du Mastermind, les répétitions peuvent être autorisées ou non, le nombre de combinaisons possible est donné par l'algorithme suivant :

Listing 1 – Algorithme de calcul du nombre de combinaisons

```
1 Si les repetitions sont autorisé alors
2
3 nombre_de_combinaisons = n^k
4
5 Sinon
6
7 nombre_de_combinaisons =  n! / (n-k)!
8
9 Finsi
```

2.2 Recherche du nombre d'indications données après chaque tentative

Nous avons également besoin de déterminer le nombre d'indications fournis après chaque tentative. Ces indications consistent pour le joueur ayant inventé la combinaison à placer autant de boules noires qu'il y a des boules bien placées dans la combinaison tenté, et de mettre autant de boules blanches qu'il y a de boules dont la couleur est présente mais qui n'est pas au bonne emplacement. L'ordre des boules noirs et blanches n'a donc aucune importance, et comme il peut y avoir plusieurs boules noires ou boules blanches de présentes les répétitions sont donc autorisées. En combinatoire, la formule correspondant à ses contraintes est :

$$R_n^k = C_{n+k-1}^k$$

Avec n le nombres d'objets distincts, c'est-à-dire 3 puisqu'il y a soit une boule noir, soit une boule blanche, soit aucune boules, et k le nombre de cases de la combinaison, on en déduit la formule :

$$\begin{aligned} nb\_indications &= R_3^k \\ &= C_{3+k-1}^k \\ &= C_{k+2}^k \\ &= \frac{(k+2)!}{k! \times (k+2-k)!} \\ &= \frac{(k+2)!}{(k! \times 2)} \end{aligned}$$

Remarque : A la formule précédente, il est nécessaire de soustraire 1 indication. Car il y a toujours une indication qu' il n'est pas possible d'avoir. Il s'agit de l'indication donnant k-1 boules noires et 1 boules blanche. Ce qui signifie qu'il y a k-1 boules bien placés et 1 boules mal placé. Or s'il y a k-1 boules bien placées la dernière boule est soit de la mauvaise couleur, soit de la bonne couleur **et** bien placée, puisqu'elle ne pourrait pas être placée ailleurs comme les autres boules sont bien placées. La formule exacte est donc :

$$nb\_indications = \frac{(k+2)!}{(k! \times 2)} - 1$$

2.3 Calcul du nombre minimum de questions à poser "dans le pire des cas"

Nous venons de trouver le nombre de combinaisons possibles et d'indications données à chaque tentative. Nous pouvons désormais en déduire le nombre minimum de questions à poser "dans le pire des cas" grâce à la formule de a Shanon :

$$nb\_tentatives\_dans\_le\_pire\_des\_cas = \lceil (\frac{\log_2(nb\_combinaison)}{\log_2(nb\_indication)}) \rceil$$

donc lorsque les répétitions de boules de couleurs sont autorisées :

$$nb\_tentatives\_dans\_le\_pire\_des\_cas = \lceil (\frac{n^k}{\frac{k+2)!}{(k! \times 2)-1)}}) \rceil$$

et lorsqu'elles sont interdites :

$$nb\_tentatives\_dans\_le\_pire\_des\_cas = \lceil (\frac{\log_2(\frac{n!}{(n-k)!})}{\log_2(\frac{(k+2)!}{(k!\times 2)} - 1)}) \rceil$$

Effectuons le calcul du nombre de tentatives dans le pire des cas, selon la règle de Mastermind la plus utilisée :

- 8 boules de couleurs (N=8)
- 4 cases (K=4)
- Répétitions autorisée

Calcul du nombre de combinaisons (sachant que les répétitions de couleurs sont autorisées) :

$$\begin{aligned} nombre\_de\_combinaisons &= n^K \\ &= 8^4 \\ &= 4096 \end{aligned}$$

Il y a 4096 combinaisons possible avec ces règles, déterminons le nombre d'indications à chaque tentatives :

Calcul du nombre d'indications :

$$\begin{aligned} nb\_indications &= R_3^k \\ &= \frac{(k+2)!}{(k! \times 2)} - 1 \\ &= \frac{6!}{(4! \times 2)} - 1 \\ &= 6 \times \frac{5}{2} - 1 \\ &= 15 - 1 \\ &= 14 \end{aligned}$$

Nous avons donc 14 indications possibles avec ces règles, calculons maintenant le nombre de tentatives dans le pire des cas :

$$\begin{aligned} nb\_tentatives\_dans\_le\_pire\_des\_cas &= \lceil (\frac{\log_2(nombre\_de\_combinaisons)}{\log_2(nombre\_d'indications)}) \rceil \\ &= \lceil (\frac{\log_2(4096)}{\log_2(14)}) \rceil \\ &= \lceil (\frac{12}{3.8074}) \rceil \\ &= \lceil (3.2) \rceil \\ &= 4 \end{aligned}$$

Le nombre minimum de questions à poser "dans le pire des cas" est donc 4.

Si vous êtes habitué à résoudre des Mastermind, ce nombre doit, comme pour nous, vous paraître très faible. En effet, comme nous la confirmé Emmanuel SAPIN[1], il s'agit ici d'un nombre "purement théorique", que les meilleurs stratèges du Mastermind tentent d'approcher.

Nous avons voulu cependant savoir pourquoi ce nombre est si faible. Et en analysant de plus près le nombre d'indications fournis à chaque tentatives on se rend compte que même si en théorie nous avons R_3^k indications possibles, en pratique le nombre d'indications est plus faible.

Exemple, lorsqu'on tente une combinaison avec que des boules rouges. Il ne peut pas y avoir d'indications comportant de boules "mal placées", puisque si des boules rouges sont présentes dans la combinaison a trouvé, alors elles seront comptées en "bien placées". Le nombre d'indications possibles dans ce cas là est donc égale aux nombre de case soit k, ce qui est bien plus faible que R_3^k. Ce constat explique en partie pourquoi il est si dur d'atteindre un nombre maximum de 4 tentatives dans le pire des cas, pour un Mastermind 8 couleurs, 4 cases.

3 Résolution de Mastermind en Algorithmique classique

Notre première approche de résolution d'une combinaison, consiste à programmer en algorithmique classique, un code capable de réaliser le même raisonnement que celui d'un humain lors de la recherche d'une combinaison de Mastermind.

Dans cette partie nous verrons comment nous sommes arrivé à cet objectif, en détaillant les critères heuristiques utilisés ainsi que les différentes fonctions d'optimisation implémentés.

3.1 Modélisation du problème du Mastermind en algorithmique classique

Le Mastermind fait partie des jeux de réflexions modélisable sous forme d'un arbre de possibilité. Cependant avec les règles habituelles de Mastermind l'arbre des possibilités à une taille beaucoup trop grande pour être exploitable à la main. Afin de nous facilité la recherche de critères heuristiques nous nous sommes donc tourné vers des arbres de possibilités de plus petites tailles grâce à des règles de Mastermind réduite (2 cases, 3 couleurs). Les critères heuristiques définis ici, sont le fruit de la réflexion sur ces arbres de possibilités.

Le principe général de résolution d'une combinaison consiste à déterminer la position de chaque couleur les unes après les autres jusqu'à ce qu'on ait trouvé la combinaison secrète. On rappelle que l'algorithme doit être capable de résoudre les Masterminds contenant des répétitions de la même couleur. Par conséquent, la première information qu'on doit obtenir est le nombre d'occurrences d'une couleur, la seconde est la position exacte de chaque occurrence dans la combinaison.

Nous en avons donc déduits les deux critères suivants :

- **Critère 1** : Trouver le nombre d'occurrences d'une couleur. Pour cela on remplit la combinaison à tenter de la couleur recherchée. Le nombre de couleurs bien placées fourni en retour moins

[1]Emmanuel SAPIN est un chercheur en informatique qui a obtenu la 3ième place à un concours d'algorithmique visant à résoudre un Mastermind 13 couleurs, 13 cases.

le nombre de boules que nous avons déjà correctement placé, correspond alors au nombre d'occurrences de la couleur :

- **Critère 2** : Trouver la position exacte de chaque occurrence de couleur. Pour cela ont teste chaque position avec la couleur en cours et on rempli le reste de la combinaison tenté avec la couleur suivante. Lorsque le nombre de couleurs mal placées fourni en retour est égale à 0, on à trouvé la position de la couleur.

Détaillons un exemple de résolution d'une combinaison appliquant ces deux critères, en résolvant la combinaison 1 3 2 1 5 :

1. On cherche le nombre d'occurrence de la couleur 1 (critère 1) :
   - Tentative 1 : 1 1 1 1 1
   - Résultat : bien_place = 2 mal_place = 0
   - $\rightarrow$ On en déduit q'il y a deux fois la couleur 1 dans la combinaison secrète (critère 1).
2. On cherche ensuite à déterminer la place exacte du premier 1. Pour cela, on affecte 1 à la première valeur et 2 aux autres valeurs. Dès que le test ne renvoi aucun pion mal placés on a trouvé la position du 1 (critère 2) :
   - Tentative 2 : 1 2 2 2 2
   - Résultat : NbBienPlace = 2 NbMalPlace = 0
   - $\rightarrow$ On a aucune couleur mal placé, par conséquent la couleur 1 est à la bonne position. On peut fixe alors la position du 1 comme sûr.
3. On détermine ensuite la position du second 1 de la même manière, sans toucher au premier 1 :
   - Tentative 3 : 1 1 2 2 2
   - Résultat : NbBienPlace = 2 NbMalPlace = 1
   - $\rightarrow NbMalPlace > 0$, la position testée n'est donc pas la bonne.
4. On test la position suivante (critère 2) :
   - Tentative 4 : 1 2 1 2 2
   - Résultat : NbBienPlace = 1 NbMalPlace = 2
   - $\rightarrow NbMalPlace > 0$, la position n'est toujours pas la bonne.
5. On test la position suivante (critère 2) :
   - Tentative 5 : 1 2 2 1 2
   - Résultat : NbBienPlace = 3 NbMalPlace = 0
   - $\rightarrow NbMalPlace = 0$, ont à trouvé la bonne position.
6. On détermine de la même manière les occurrences des autres couleurs à la différence près que le nombre de bien placés est égal au nombre de pions bien placés moins le nombre de pions déjà trouvés (critère 1) :
   - Tentative 6 : 1 2 2 1 2
   - Résultat : bien_place = 3 mal_place =0
   - $\rightarrow$ Il y a 3-2=1 boule de couleur 2.
7. Recherche de la position de la boule de couleur 2 (critère 2) :
   - Tentative 7 : 1 2 3 1 3
   - Résultat : bien_place = 3 mal_place = 3

- $\rightarrow NbMalPlace > 0$, la position n'est pas la bonne (critère 2)

8. On continu la recherche de la position :
   - Tentative 8 : 1 3 2 1 3
   - Résultat : bien_place = 4 mal_place =0
   - $\rightarrow NbMalPlace = 0$, ont à trouvé la bonne position.
9. On cherche le nombre d'occurrences de la couleur 3 (critère 1) :
   - Tentative 9 : 1 3 2 1 3
   - Résultat : bien_place = 4 mal_place = 0
   - $\rightarrow$ Il y a 4-3=1 boule de couleur 3.
10. On cherche la position de la boule de couleur 3 (critère 2) :
    - Tentative 10 : 1 3 2 1 4
    - Résultat : bien_place = 4 mal_place = 0
    - $\rightarrow NbMalPlace = 0$, ont à trouvé la bonne position.
11. On cherche le nombre d'occurrences de la couleur 4 (critère 1) :
    - Tentative 11 : 1 3 2 1 4
    - Résultat : bien_place = 4 mal_place = 0
    - $\rightarrow$ Il y a 4-4=0 boule de couleur 4, on passe à la couleur 5.
12. On cherche le nombre d'occurrences de la couleur 5 (critère 1) :
    - Tentative 12 : 1 3 2 1 5
    - Résultat : bien_place = 5 mal_place = 0
    - $\rightarrow$ Il y a 5-4=1 boule de couleur 5, ET bien_place = 5.
      La combinaison à été trouvées.

Nous venons de voir qu'en implémentant seulement les deux critères précédents, nous pouvons écrire un algorithme capable de résoudre une combinaison. C'est d'ailleurs ce que nous avons fait avec les stratégies 1 et 2.

Cependant on remarque que la recherche de la position de chaque occurrence de couleur demande beaucoup de tentatives. Afin de discriminer encore un peu plus les possibilités nous nous sommes posés la question suivante : Pouvons-nous continuer à obtenir de nouvelles informations sur la combinaison lors de la recherche de la position d'une couleur ?

Après une analyse plus approfondie du problème, nous avons pu trouver des nouveaux critères heuristiques, tournant justement la longue recherche de position d'une couleur à notre avantage. Nous pouvons donc répondre par l'affirmative à la question précédente.

Explications :

- **Critère 3** : Lors de la recherche de la position exacte de la couleur courante nous complétons les cases de la combinaison non encore déterminée avec la couleur suivante (Critère 2). Il est alors possible de connaître le nombre d'occurrences de la couleur suivante, sans avoir besoin de la tester lors de la prochaine itération de la boucle. En effet, en analysant le nombre de boules bien placées et le nombre de boules mal placé on peut directement obtenir cette information par la formule :

$$Nombre\_de\_boules\_de\_la\_couleur\_suivante = (NbBienPlace + NbMalPlace) - NbDejaTrouve - 1.$$

Au nombre de boules bien placées et mal placées, on soustrait le nombre de boules de couleurs dont ont à déjà déterminé la position (NbDejaTrouve) puis on soustrait 1 qui correspond à la couleur courante dont on sait qu'elle appartient à la combinaison. Le nombre résultant est donc bien le nombre d'occurrence de la couleur suivante.

Cette formule nous permet d'éviter les tentatives permettant de déterminer le nombre d'occurrences (Critère 1). Puisque quand la couleur suivante deviendra la couleur courante, nous connaîtrons déjà à l'avance son nombre d'occurrences.

- **Critère 4** : Puisqu'avec le Critère 3 nous connaissons le nombre d'occurrences de la couleur prochainement testé, nous pouvons savoir si cette couleur est présente ou non. Il suffit en effet de regarder si son nombre d'occurrence vaut zéro. Si c'est le cas, au lieu de continuer à compléter la combinaison avec cette couleur, on complète la combinaison avec la couleur supérieure.

L'implémentation de ces quatre critères dans la stratégie 3, nous permet d'avoir une stratégie de résolution aussi intelligente que le raisonnement humain. Les nombres de tentatives moyens et dans le pire des cas ainsi que le temps d'exécution ont alors été grandement diminués, comme nous le verrons dans la partie "Comparaisons des performances des stratégies implémentées".

3.2 Schéma d'algorithme des implémentations

Nous avons réalisé trois stratégies implémentant les critères précédents. Les deux premières implémentent les critères 1 et 2 mais avec une approche différente au niveau algorithmique, et la troisième implémente tous les critères heuristiques.

3.2.1 Stratégies implémentant les critères 1 et 2

Les stratégies expliquées dans cette partie implémentent les deux critères minimums pour résoudre une combinaison de Mastermind. Elles nous ont permis de comprendre les principales difficultés d'implémentations. Nous avons ensuite gardé le meilleur de ces deux stratégies pour réaliser la stratégie 3 implémentant les deux autres critères.

Le schéma d'algorithme de ces deux stratégies est le suivant :

Listing 2 – Pseudo-code des stratégies 1 et 2

```
1  Initialiser la couleurCourante à 1
2
3  Tant que le Mastermind n'est pas résolu Faire
4
5    – [Critère 1] Générer une tentative cherchant la présence le nombre d
       'occurence de la couleurCourante
6
7    – Soumettre la tentative au programme qui à générer la combinaison
       pour récupérer le nombre de boule bien placé et mal placé
8
9    – [Critère 1] Déterminer le nombre d'occurence de la couleurCourante
       par la formule : nbOccurence = nbBienPlace – nbBouleDejaBienPlace
10
```

```
  - Si la couleurCourante est présente dans la combinaison

        - Pour chaque occurrence de la couleur courante

            - [Critère 2] Tant que le nombre de boule mal placé est
              différent de 0

                - [Critère 2] Générer une tentative cherchant la position
                  exacte de la couleurCourante

            - Fin Tant que

        - Fin Pour

  - Fin Si
  couleurCourante <- couleurCourante + 1

Fin Tant que
```

Stratégie 2 : Deuxième implémentation des critères 1 et 2

L'approche au niveau algorithmique de cette stratégie diffère, avec la stratégie 1, notamment avec un système de drapeau permettant de savoir si la position d'une couleur à été trouvée. Ce qui permet à cette stratégie d'obtenir de meilleurs résultats. Cependant le schéma d'algorithme de cette stratégie est le même que pour la stratégie 1, afin d'appréhender leur différence nous vous invitons à consulter l'annexe 2 détaillant le pseudo-code détaillé des stratégies implémentés.

3.2.2 Stratégie implémentant les quatre critères heuristiques

Cette partie est consacrée à la troisième stratégie que nous avons programmé. Celle-ci implémente tous les critères heuristiques définis précédement et cela par un algorithme le plus optimisé en nombre de lignes et en temps d'exécution, comme nous le verrons plus tard dans la partie consacrée aux tests. Les tentatives qu'elle propose pour résoudre une combinaison sont équivalentes aux tentatives tentées par un humain. Il s'agit là d'une des meilleures stratégies possibles en algorithmique classique.

Le schéma d'algorithme de cette stratégie est le suivant :

Listing 3 – Pseudo-code de la stratégie 3

```
Initialiser la couleurCourante à 1

Tant que le Mastermind n'est pas résolu Faire

  - [Critère 1] Générer une tentative cherchant la présence le nombre d
    'occurence de la couleurCourante

  - Soumettre la tentative au programme qui à générer la combinaison
    pour récupérer le nombre de boule bien placé et mal placé
```

```
8
9   - [Critère 1] Déterminer le nombre d'occurence de la couleurCourante
        par la formule : nbOccurence = nbBienPlace - nbBouleDejaBienPlace
10
11  - Si la couleurCourante est présente dans la combinaison
12
13      - Faire
14
15          - Pour chaque occurrence de la couleur courante
16
17              - [Critère 2] Tant que le nombre de boule mal placé est
                    différent de 0
18
19                  - [Critère 2] Générer une tentative cherchant la position
                        exacte de la couleurCourante
20
21                  - [Critère 3] Determiner le nombre d'occurence de la
                        couleur suivante par la formule
                        nbOccurenceCouleurSuivante  = (nbBienPlace+nbMalPlace)
                         - nbTrouve - 1
22
23                  - [Critère 4] Si la couleur suivante n'est pas dans la
                        combinaison
24                                          couleurSuivante =
                                               couleurSuivante + 1
25
26              - Fin Tant que
27
28          - Fin Pour
29
30      - Tant que la couleur suivante est présente
31
32  - Fin Si
33    couleurCourante <- couleurCourante + 1
34 Fin Tant que
```

Sur un schéma d'algorithme l'implémentation de ces deux nouveaux critères peut paraître simple. Cependant l'algorithmique classique regorge de subtilités dont il a fallu faire face. Pour information l'implémentation en Java de cette algorithme tient en plus de 200 lignes de code (environ 160 en pseudo-code détaillées comme vous pourrez le constater dans l'annexe 2).

Comme nous le verrons dans la partie consacrée aux tests, l'apport des critères 3 et 4 augmentent grandement l'efficacité de l'algorithme. Et les tentatives qu'il propose sont similaires à ce qu'un être humain propose. Cependant on s'aperçois que même l'apport des critères 3 et 4 ne permettent pas d'exploiter aux maximum les informations fournies après chaque tentative. En fait ce constat est intrinsèque à l'algorithmique classique, qui est toujours dans l'attente de la satisfaction d'une contrainte avant de passer à une autre. D'où la question suivante : Est-il possible d'écrire

un programme capable à chaque instant de tenir compte de toutes les informations obtenues après chaque tentative ?

4 Modélisation du problème du Mastermind en Programmation Par Contrainte (PPC)

Nous allons maintenant voir une nouvelle approche de résolution d'une combinaison de Mastermind. Cette méthode, appelée "Programmation par contrainte", consiste à définir formellement un problème en définissant ses variables et leurs domaines ainsi qu'un ensemble de contraintes.

Ce type de programmation permet de tenir compte de toutes les informations obtenues après chaque tentative. Ce rapport se devant être compréhensible pour tout étudiant, nous avons réalisé un rappel définissant plus en détail ce qu'est la programmation par contrainte dans l'annexe 1, que nous vous invitons à consulter si vous découvrez pour la première fois cette nouvelle méthode de programmation.

4.1 Modélisation du problème du Mastermind

4.1.1 Description formel d'une combinaison

Reprenons les règles du Mastermind afin de les retranscrire en un problème de satisfaction de contrainte (CSP).

Le jeu du Mastermind consiste à déterminer un code secret composé de N valeurs (chaque valeur représentant une couleur) réparties dans K cases. Le code secret est donc défini par l'ensemble ordonné de variables $X = \{X1, X2, X3, ..., Xk\}$. Le domaine quant à lui est similaire pour chaque variable : $D = \{1, ..., N\}$

Le but est de trouver le code en proposant des combinaisons successives, basées sur un retour d'information fourni après chaque tentative par l'inventeur de la combinaison.

L'information fournie en retour est le nombre de bons chiffres à la bonne position (NbBienPlace) dans le code (sans dire quelles sont les bonnes positions) et le nombre de bons chiffres situés à des mauvaises positions (NbMalPlace) dans le code.

Les variables définissant le code secret recherché devront être représentées sous forme d'un tableau combi[] de taille K, composé de variables entières, dont les domaines seront restreints à [1 ... N].

La méthodologie de la programmation par contrainte consiste à définir un domaine de validité pour chaque variable dans le système, ainsi que des contraintes relatives entre les variables. Par la suite, il convient de fixer la valeur d'une première variable, propager les nouvelles contraintes et fixer la variable suivante, jusqu'à ce que toutes les variables aient des valeurs attribuées.

Si une contradiction entre deux contraintes est détectée (c'est à dire, si un domaine d'une variable devient "vide"), un "***backtrack***" (reprise) est effectué, afin d'essayer d'autres valeurs.

Il faut imposer de plus en plus de contraintes et restreindre ainsi successivement les domaines de chaque variable jusqu'à ce qu'il ne reste qu'une seule valeur possible. C'est à ce moment qu'on aura trouvé la bonne solution.

De plus, un autre tableau de variables entières, qu'on nommera Occ[] de taille N, sera à définir. Ce dernier servira à déterminer le nombre d'occurrences de chaque chiffre. A priori les domaines de

ces variables vont de [0 ... K], car un chiffre peut exister entre "aucune fois" et "toutes les positions sont prises par le même chiffre".

Listons les contraintes retenues pour la résolution d'une combinaison :

- Contrainte 1 : La première contrainte imposée est donc que le nombre d'occurrences d'un chiffre i doit être inclus dans le domaine Occ[i].
- Contrainte 2 : La somme des minimas des occurrences de toutes les couleurs doit être égale à la somme du nombre de chiffres bien placé et du nombre de chiffres mal placés.
- Contrainte 3 : En comparant chaque élément d'une tentative T avec la tentative précédente (T-1), on doit obtenir le même nombre de correspondances :

$$NbBienPlace(T-1 \Leftrightarrow X) = NbBienPlace(T-1 \Leftrightarrow T)$$

Explication : NbBienPlace indique le nombre de bons chiffres aux bonnes positions, lorsque la tentative T-1 est comparée à la combinaison recherchée. Forcément, pour être la bonne combinaison, une tentative T doit avoir le même nombre NbBienPlace, si l'on compare à T-1.

4.1.2 Schéma de l'algorithme

L'algorithme de propagation de contrainte du problème du Mastermind devra être implémenté comme suit :

1. Créer les tableaux combiTente de taille K et Occ de taille N, restreindre les domaines de leurs variables à [1 ... N] et [0 ... K]
2. Imposer la contrainte (C1)
3. Tant que NbBienPlace < K
   (a) Fixer une valeur quelconque pour chacune des variables du tableau combiTente[] (en respectant les domaines et les contraintes)
   (b) Proposer la tentative obtenus et récupérer NbBienPlace et NbMalPlace
   (c) Imposer les nouvelles contraintes (C2) et (C3)
4. La bonne séquence de chiffres a été trouvée

Le langage le plus adapté pour implémenter un problème modélisé en programation par contrainte est le langage Prolog. Comme il fera partie de notre enseignement au lors du semestre prochain. Nous espérons à ce moment là, pouvoir implémenter l'algorithmique donnée plus haut.

Enfin, nous tenons à remercier Sébastian Kanzow docteur en informatique à l'Université de Paris 12, pour ses précisions notamment sur les contraintes à retenir.

5 Comparaisons des performances des stratégies implémentées

5.1 Protocole de test

Afin de comparer les stratégies implémentées, nous avons réalisée un ensemble de tests. Dont voici le protocole :

Machine de test :

– Processeur : 3,5ghz (AMD Athlon core Venice)
– Mémoire : 2 Giga de RAM (Corsaire)
– Disques durs : 2 x 250Go, 7200tr/min (Maxtor diamond max 10)

Nombres de mastermind résolu pour chaque test : 1 000 000

Chaque stratégie à été testée avec cinq règles différentes, afin de voir si une stratégie est meilleure que les autres en fonction des règles de Mastermind.

Les cinq règles testées sont les suivantes :

1. Règles 1 :
   – 6 couleurs
   – 4 cases
   – Répétitions autorisées

2. Règles 2 :
   – 8 couleurs
   – 4 cases
   – Répétitions autorisées

3. Règles 3 :
   – 8 couleurs
   – 4 cases
   – Pas de répétition de couleur

4. Règles 4 :
   – 20 couleurs
   – 4 cases
   – Répétitions autorisées

5. Règles 5 :
   – 4 couleurs
   – 6 cases
   – Répétitions autorisées

5.2 Graphique statistiques

5.2.1 Nombre de tentatives moyen

Dans ce premier graphique nous pouvons voir le nombre moyen de tentatives réalisées par chaque stratégie pour résoudre une combinaison.

Premièrement on peut remarquer que la stratégie 2 est légèrement meilleure que la stratégie 1, mais qu'en règle générale les deux stratégies obtiennent des résultats très similaires. Ce constat s'explique facilement puisque les stratégies 1 et 2 implémentent toutes les deux les mêmes critères heuristiques. Cependant lorsque que la combinaison à rechercher contient plus de cases que de

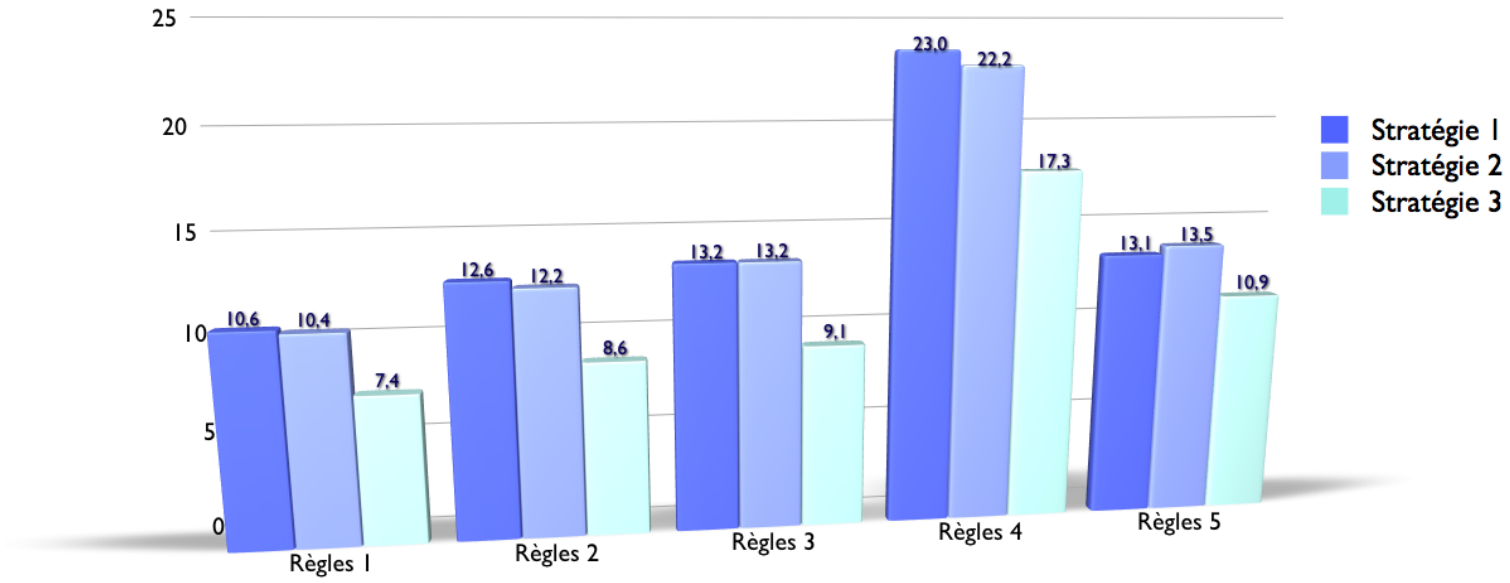

FIG. 4 – Nombre de tentatives moyen selon les trois stratégies et selon différentes règles de Mastermind

couleurs c'est la stratégie 1 qui réalise moins de tentatives en moyenne. Nous expliquerons pourquoi nous faisons ce constat, un peu plus loin.

Le stratégie 3 quant à elle obtient d'excellents résultats avec un gain moyen de 30% par rapport aux deux premières stratégies. On peut donc affirmer dès maintenant que l'implémentation des critères 3 et 4 nous a permis d'améliorer de manière importante l'heuristique de notre algorithme.

5.2.2 Nombre de tentatives maximum

Ce second graphique Fig.[5] met en avant le nombre de tentatives effectuées dans le pire des cas, pour résoudre une combinaison selon les trois stratégies.

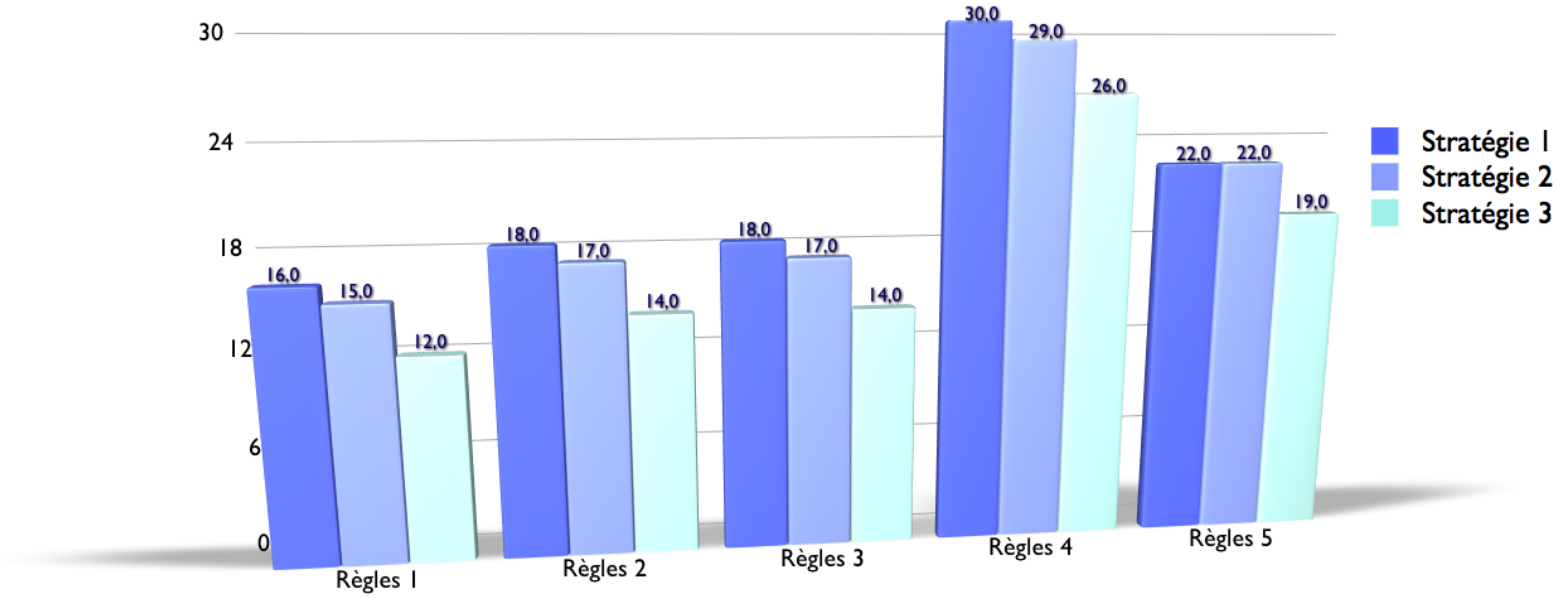

FIG. 5 – Nombre de tentatives maximum selon les trois stratégies et selon différentes règles de Mastermind

Ce nombre est très parlant pour arriver à juger l'intelligence d'une stratégie. La aussi on obtient le classement suivant :

1. Stratégie 3
2. Stratégie 2
3. Stratégie 1

5.2.3 Temps d'execution

Examinons maintenant les temps d'exécution. Ces temps sont exprimés en milisecondes, plus le temps d'exécution est faible, meilleure est la stratégie.

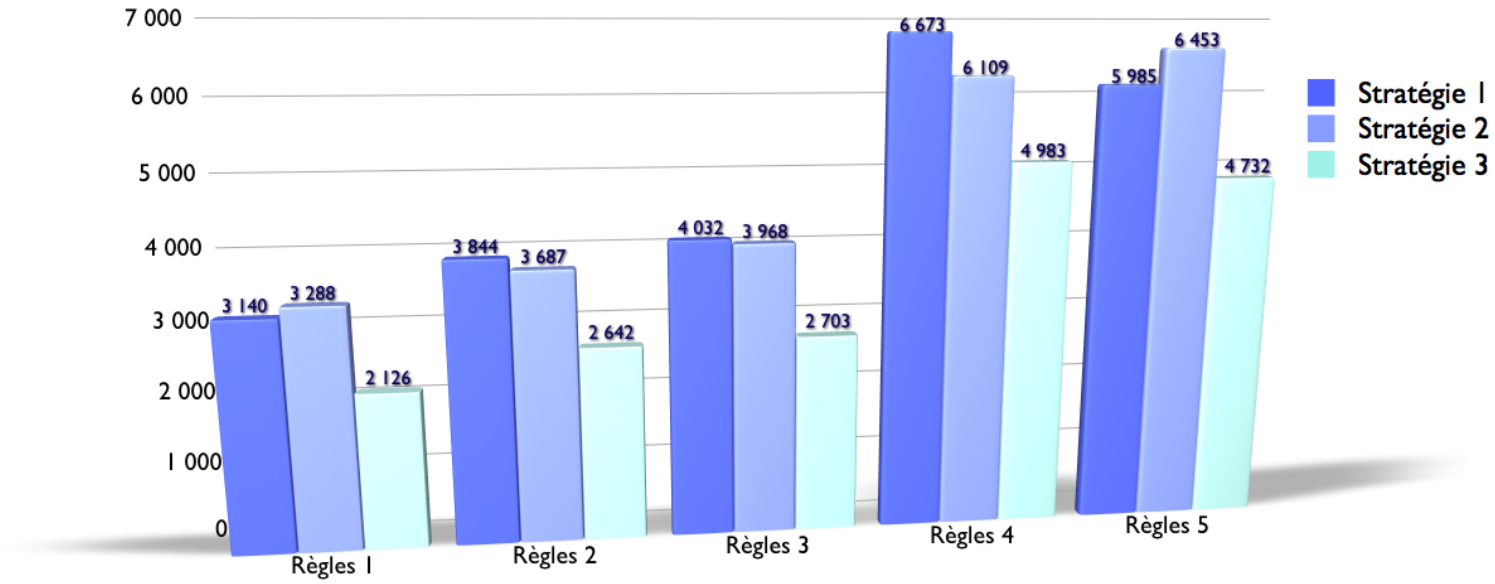

FIG. 6 – Temps d'execution total en milisecondes pour résoudre 1 000 000 de combinaisons, selon les trois stratégies et selon différentes règles de Mastermind

On remarque là encore que la stratégie 3 obtient de très bons résultats avec un gain moyen de 35 % de rapidité par rapport aux stratégies 1 et 2, ce qui est logique puisqu'elle est capable de résoudre une combinaison en moins de tentatives que ces deux dernières.

On peut également remarquer que nos impressions sur le nombre de tentatives moyen de la stratégie 2 pour résoudre les Mastermind 4 couleurs, 6 cases (Règles 5) sont confirmées. Cette stratégie obtient de moins bons résultats que la Stratégie 1 pourtant moins efficace avec toutes les autres règles. On en conclu que la recherche de position (Critère 2) de la stratégie 2 est moins bien réalisé que dans la stratégie 1, car la règle 5 nécessite plus souvent que les autres, la recherche de la position d'une couleur.

5.3 Interface de mise en concurrences des stratégies

Afin de facilité la démonstration de nos algorithmes le jour de la démonstration, nous avons programmer une interface graphique (Fig. [7]). Cette interface nous a également permis de réaliser nos différents testes, et comme nous avons encapsulé chaque stratégie dans un processus (Thread) différent, nous pouvons lancer les trois stratégies en parallèle et voir ainsi en temps réelle celle qui résout le lot de combinaisons le plus rapidement possible. Nous pouvons ainsi mettre en avant de manière plus conviviale les différences entre les stratégies.

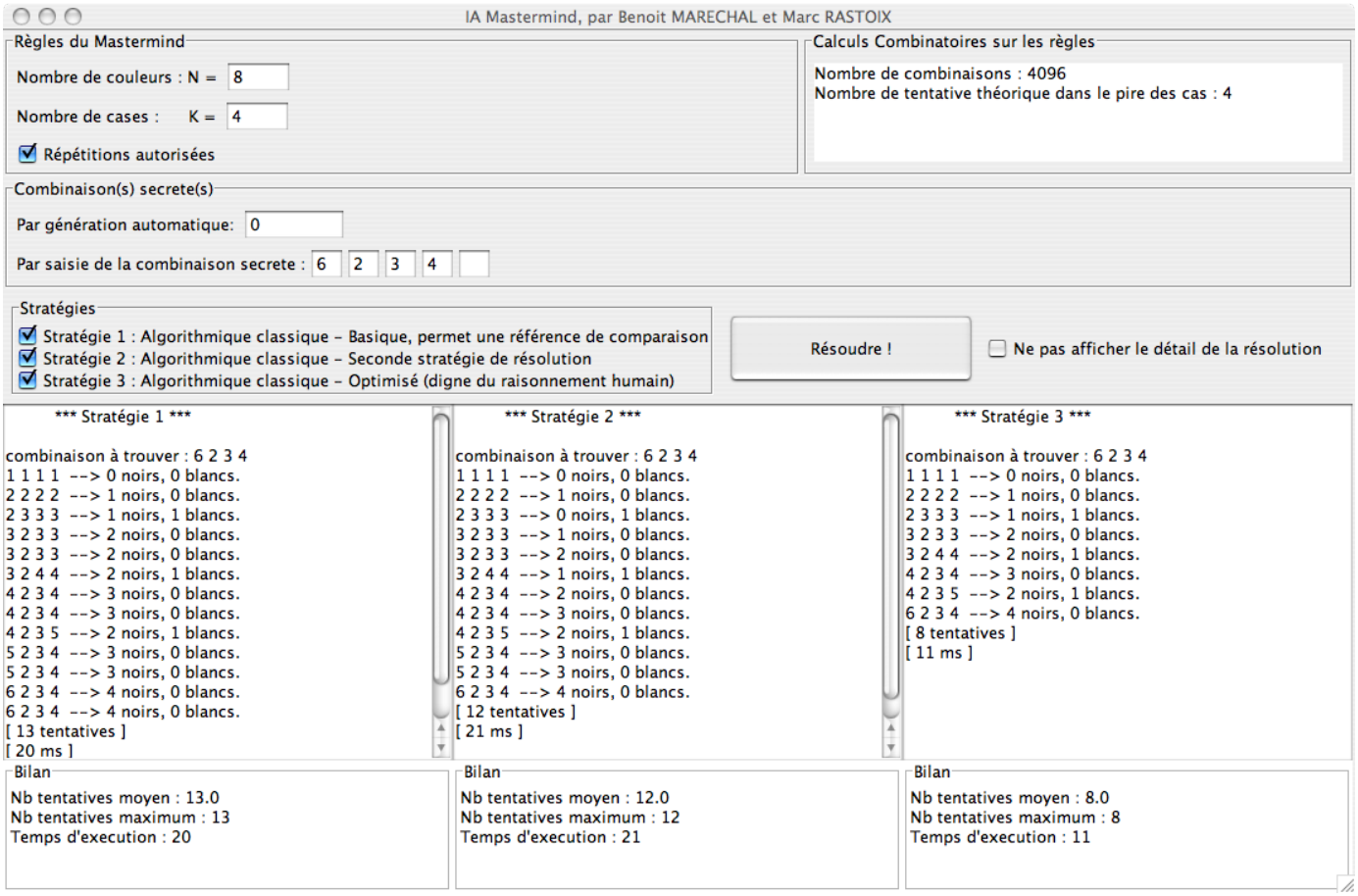

FIG. 7 – Interface graphique de mise en concurrence des stratégies

6 Conclusion

Même si au premier abord, travailler sur l'intelligence artificielle d'un jeux de réflexion paraît simple, en réalité il n'en est rien. Comme vous pourrez le constater dans l'annexe 2 consacré au pseudo-code des algorithmes. La moindre stratégie de résolution, demande plus d'une centaine de lignes de pseudo-code. Il est donc évident qu'une modélisation et une analyse précise du problème doivent être réalisées avant de commencer les implémentations. C'est pourquoi nous avons effectué en premier lieu une investigation mathématique sur le mastermind, ainsi qu'une recherche de critères nous permettant d'élaguer le plus rapidement possible l'arbre des possibilités. Nous avons également pu constater que la programmation en algorithmique classique, pour ce genre de problème, montre rapidement ses limites. Tandis que l'approche par propagation de contrainte, elle, ne souffre pas de toutes ces difficultés d'implémentation et elle permet en plus de tenir compte à chaque instant de toutes les informations dont elle dispose, contrairement à l'algorithmique classique. C'est d'ailleurs pourquoi, cette approche est considérée comme la plus performante pour résoudre un grand nombre de jeux de réflexions, mais également de problèmes plus concret du monde professionnel. S'orientant vers un Master Intelligence artificielle et base de donnée, nous espérons que nous continuerons à approfondir ce type de programmation très prometteur.

7 Bibliographie

Ce rapport a été réalisé à partir des ouvrages documents suivants :

Références

[1] Cormen, Leiserson, Rivest. DUNOD - 2002 - Livre : Introduction à l'algorithmique

[2] Jacques Courtin, Irène Kowarski - DUNOD - 1994 - Livre : Initiation à l'algorithmique et aux structures de données

[3] Christine Solnon (1997) http://bat710.univ-lyon1.fr/~csolnon/contraintes.html

[4] Sébastien Dorane - http://sjrd.developpez.com/algorithmique/DidierDeleglise

[5] Code Source France : http://www.cppfrance.com/codes/source_37316.aspx

[6] Concours Mastermind organiser entre scientifiques - Université Paris V René Descarte - http://www.antsearch.univ-tours.fr/ea/default.asp?FCT=DP&ID_PAGE=24