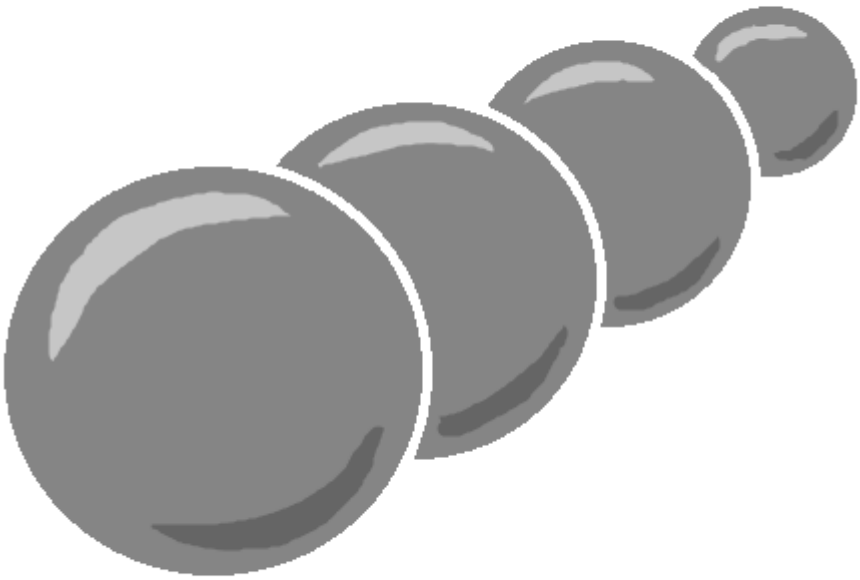
Mastermind

Un jeu compliqué ?



Auteur : Sidney Barthe

Maître responsable : Didier Müller
Expert : Olivier Dubail

Index

Annexe :
- CD-ROM *Mastermind* comprenant :
    - *mastermind_stats.py*
    - *mastermind_jeu.py*
    - *mastermind.odt*
    - *mastermind.pdf*
    - *python-2.6.4.msi*
    - dossier *images*

1. Introduction

J'ai décidé, dans le cadre de mon travail de maturité en informatique, comme suggéré par Didier Müller, d'étudier un jeu bien connu qu'est le Mastermind. J'ai choisi ce sujet, car je m'intéresse aux jeux de logique et je dois avouer que je n'ai jamais été fort à ce jeu et que j'ai souvent eu des problèmes pour comprendre comment gagner et comment faire les déductions qui amènent à la victoire. De plus, ayant toujours eu de l'intérêt et certaines notions en informatique, j'ai trouvé que c'était une bonne idée de l'utiliser pour comprendre ce jeu. L'informatique est une science qui est désormais omniprésente dans la vie de tous les jours et il s'avère que c'est également un outil extrêmement puissant pour résoudre tous les jeux de logique et autres casse-têtes qui ont torturé l'humanité depuis longtemps. Je trouve cela étonnant comme un joueur d'échecs ou une grille de sudoku peuvent être ridiculisés par un ordinateur.

1.1. Le Mastermind

Présentation

Le Mastermind est un jeu de société, de réflexion et de déduction, se jouant à deux joueurs, inventé en 1962 par Mordecai Meirovitz. Il s'agit d'une table composée d'un certain nombre de rangées horizontales de petits trous servant à y placer des boules ou des pions de couleurs.

Déroulement

Le jeu se joue à 2 joueurs. Un joueur doit au départ remplir une ligne cachée pour l'autre joueur avec des boules en utilisant les couleurs de son choix. L'autre joueur doit deviner la bonne combinaison en proposant successivement, avec un nombre limité d'essais, une combinaison de son choix. Le joueur connaissant la solution doit fournir à son adversaire des informations à chaque fois qu'il propose une combinaison. Il doit lui indiquer le nombre de boules correctes, c'est-à-dire de bonne couleur et bien placées (symbolisé par des pions noirs, ou rouge selon les versions, que l'on place à côté de la proposition), et le nombre de boules mal placées (symbolisé par des pions blancs). En aucun cas il n'indique quelles sont les boules bien placées ou mal placées.

Fin de partie

Il peut alors se produire deux situations : le joueur a réussi à trouver la bonne combinaison (celle que l'autre joueur a inventée). Dans ce cas, il gagne. S'il ne l'a pas trouvée après un certain nombre d'essais (défini par le nombre de rangées qu'il y a), c'est l'autre joueur qui gagne.

1.2. Les objectifs du travail

Dans mon travail de maturité, il s'agira de réaliser deux programmes en utilisant Python, langage de programmation assez récent et multiplate-forme. Ensuite, je pourrai les utiliser pour dresser des statistiques.

Dans mon premier programme, que je nommerai *mastermind_stats*, il s'agira de simuler des parties de Mastermind en remplaçant les deux joueurs par de l'intelligence artificielle. C'est-à-dire qu'il s'agira de parties ordinateur contre ordinateur, ceci dans le but d'effectuer énormément de parties à une vitesse accélérée tout en gardant en mémoire les données qui me seront importantes à l'étude.

Le second programme, que je nommerai *mastermind_jeu*, sera un programme dans lequel l'utilisateur pourra participer aux parties. Il s'agira de deviner la combinaison générée aléatoirement, avec si désiré, l'assistance de l'ordinateur. Ce programme servira aussi à montrer l'efficacité de la méthode que l'ordinateur utilise pour trouver la bonne combinaison.

Les statistiques porteront sur la moyenne du nombre de coups nécessaires pour découvrir la combinaison cachée, le nombre de coups maximum, le nombre de coups minimum, le nombre de possibilités éliminées par déduction à chaque coup, la recherche de la combinaison la plus difficile et la plus facile à trouver.

1.3. Glossaire

Afin que vous puissiez me comprendre plus facilement, j'ai trouvé utile de créer un petit glossaire pour comprendre immédiatement les termes qui reviendront souvent dans ce travail.

Les *boules* sont les éléments que l'on place sur les rangées du jeu afin de former des *combinaisons*. Ces boules peuvent être de plusieurs couleurs.

Les *pions* sont les éléments noirs ou blancs (rouge ou blancs selon d'autres versions) qu'on place à côté des combinaisons et qui servent à indiquer si les boules sont bien placées ou mal placées.

La *solution* est la combinaison cachée qu'un joueur doit découvrir.

Les *propositions* sont les combinaisons qu'un joueur va proposer à chaque tour pour obtenir des informations sur la solution.

Les *résultats* sont les séries de pions qui servent à indiquer la différence entre deux combinaisons (par exemple une proposition avec la solution).

Le *nombre de couleurs* est le nombre de couleurs que peuvent avoir les boules (normalement 6). Il ne faudra pas confondre le nombre de couleurs avec la *variété de couleurs* qui sera expliquée plus loin.

Les *possibilités* sont les solutions potentielles. Elles sont gardées en mémoire et diminuent en nombre grâce aux résultats.

Le *nombre de coups* est égal au nombre de propositions soumises par le joueur qui doit trouver la solution.

2. Méthode de résolution

2.1. Par l'informatique

Je vais expliquer ici pas à pas le cœur du fonctionnement de mes programmes. Il s'agit de la méthode que va employer l'ordinateur pour déduire la combinaison cachée, c'est-à-dire le moyen optimal de gagner une partie de Mastermind. Il s'agit d'une méthode par élimination.

Notez qu'à tout moment, si une étape paraît mal expliquée ou si vous voulez obtenir des détails techniques, il vous sera possible de consulter le code, mais il faut tout de même quelques notions en programmation pour comprendre.

Il faut comprendre que, puisque l'ordinateur doit jouer contre lui-même, il commence d'abord par générer une combinaison aléatoire (ou avec un paramètre que nous verrons plus loin). Cette combinaison représente bien évidemment la solution qui est cachée et qu'un joueur doit découvrir.

Étape 1 : Dresser la liste des possibilités

Il faut tout d'abord dresser une liste de toutes les combinaisons possibles en fonction du nombre de boules et de couleurs à disposition. Ainsi, on est certain que la combinaison cachée en fait partie. Notons que le nombre de possibilités est (selon le théorème des arrangements avec répétitions) : $p = n^b$

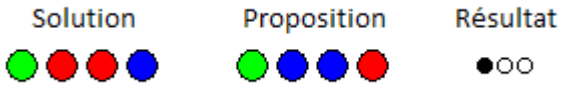
p : nombre de possibilités
n : nombre de couleurs
b : nombre de boules

Ainsi, dans le Mastermind classique (4 boules et 6 couleurs), il y a :
$6^4 = 1296$ possibilités.
Éventuellement (selon d'autres versions avec 4 boules et 8 couleurs) :
$8^4 = 4096$ possibilités.

Dans mon programme, la liste des possibilités qui est nommée *listePos* est créée dans la fonction *creerListePos*. Toutes les possibilités y sont codées sous forme de liste de chiffres (*listePos* serait alors en fait un tableau), chaque chiffre représentant une couleur. Elles sont triées par ordre numérique croissant.
(c.f. *mastermind_stats.py* lignes 37-55)

Étape 2 : Proposer une combinaison et la comparer avec la solution
Rappelons que l'ordinateur a deux rôles. Il doit non seulement trouver le code secret, mais il doit aussi se donner les informations nécessaires pour le trouver, car il est le seul à le connaître. Cela peut paraître paradoxal, mais on peut imaginer qu'il s'agit de deux intelligences artificielles qui interagissent. L'ordinateur va donc proposer une combinaison choisie aléatoirement dans la liste créée à l'étape précédente. Il doit ensuite comparer le code caché avec cette proposition pour savoir le nombre de boules bien placées et de bonne couleur (pions noirs) ainsi que le nombre de boules mal placées, mais de bonne couleur (pions blancs). Cette opération est effectuée avec la fonction *comparer*. On verra dans la prochaine étape que cette fonction servira également à autre chose. (c.f. *mastermind_stats.py* lignes 6-27)

Exemple :
Si la solution est vert-rouge-rouge-bleu et que l'on propose vert-bleu-bleu-rouge, on obtient 1 pion noir et 2 pions blancs. Le vert est placé au même endroit, alors qu'une boule rouge et une boule bleue sont placées ailleurs.



Étape 3 : Interpréter le résultat et éliminer des possibilités
Une fois la solution comparée avec la combinaison, l'ordinateur doit interpréter le résultat et faire des déductions logiques. Il va éliminer des possibilités de la liste créée à l'étape 1 grâce aux résultats.

Tout le monde comprendra que, tout d'abord, si la proposition n'est pas la solution, on peut l'éliminer de la liste.

Ensuite, il faut enlever des possibilités en fonction du nombre de pions noirs (boules de bonne couleur et bien placées). On peut éliminer toutes les possibilités qui n'ont pas exactement *N* boules placées au même endroit et de même couleur que la proposition, *N* étant le nombre de pions noirs obtenus.

Exemple :
Admettons que l'on propose bleu-rouge-vert-vert et que l'on obtienne 1 pion noir. On peut alors éliminer toutes les combinaisons qui n'ont pas :
Une boule bleue dans le premier emplacement.
ou
Une boule rouge dans le deuxième emplacement
ou
Une boule verte dans le troisième emplacement
ou
Une boule verte dans le quatrième emplacement

Notons que les combinaisons qui ne répondent pas à plus de *N* de ces conditions (ici une) doivent être éliminées. C'est-à-dire que les combinaisons avec une bleue au premier emplacement et une rouge au deuxième emplacement par exemple doivent aussi être effacées.

Pour finir, il reste à enlever des possibilités en fonction du nombre de pions blancs (boules de bonne couleur mais mal placées). On peut éliminer toutes les possibilités qui n'ont pas exactement *N* boules de même couleur mais placées ailleurs, *N* étant le nombre de pions blancs obtenus.

Exemple :
Si l'on propose bleu-jaune-jaune-rouge et qu'on obtient 1 pion blanc, on peut éliminer toutes les combinaisons qui n'ont pas :
Une boule bleue dans le deuxième, troisième ou quatrième emplacement
ou
Une boule jaune dans le premier ou quatrième emplacement
ou
Une boule rouge dans le premier, deuxième ou troisième emplacement

Ici aussi, notons qu'il s'agit de « ou » exclusifs, c'est-à-dire que les combinaisons qui ne répondent pas exactement à *N* de ces conditions (ici une) doivent être éliminées.

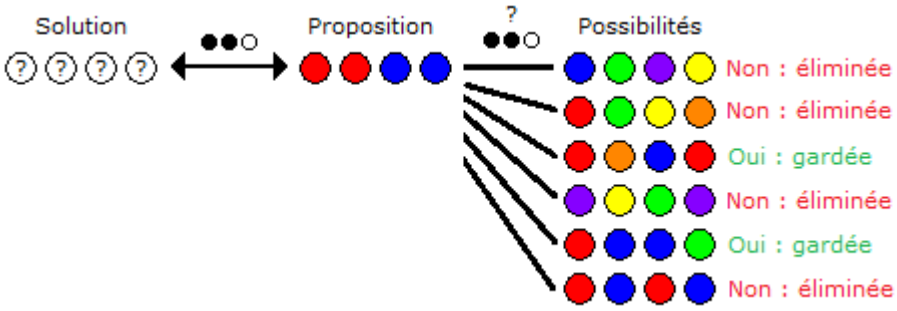
On pourrait croire qu'effectuer toutes ces éliminations demande beaucoup d'opérations. C'est d'ailleurs ce que j'ai pensé au premier abord alors que j'avais commencé à programmer plusieurs fonctions effectuant toutes ces opérations. C'est alors qu'on remarque une méthode bien plus facile : il s'agit de réutiliser la fonction *comparer*. Cette fonction a été créée initialement pour comparer la solution avec une proposition, mais en fait, on peut comparer deux combinaisons quelconques.

Il faut comprendre que la fonction est réciproque. Si l'on propose une combinaison et, qu'après la comparaison avec la solution, on obtient un certain nombre de pions blancs et de pions noirs, on obtiendrait le même résultat en inversant les rôles. J'utilise alors l'expression : « La combinaison 1 a *N* pions noirs et *B* pions blancs avec la combinaison 2. ». Cela veut dire que *N* boules sont pareilles et *B* boules sont de même couleur, mais placées ailleurs.

Exemple :
combinaison 1 : jaune-vert-vert-bleu
combinaison 2 : vert-bleu-vert-rouge
La combinaison 1 a 1 pion noir et 2 pions blancs avec la combinaison 2 et inversement.

Donc, cela veut dire que, si avec une proposition, on obtient *N* pions noirs et *B* pions blancs, la solution doit aussi avoir *N* pions noirs et *B* pions blancs avec notre proposition.

Le programme va alors, après avoir récupéré le résultat, comparer sa proposition avec toutes les possibilités de la liste. Toutes les possibilités qui, une fois comparées avec la proposition, n'ont pas un résultat égal à celui de la proposition comparée avec la solution doivent être effacées de la liste.
(c.f. *mastermind_stats.py* lignes 57-70)



Étape 4 : retour à l'étape 1

Le programme retourne à l'étape 1. C'est ainsi qu'à chaque coup, la liste des possibilités diminue en éléments. Le programme travaille donc de plus en plus vite, car il est plus facile de manier une petite liste. Il trouve donc la solution lorsqu'il obtient autant de pions noirs qu'il y a de boules.

2.2. « Humainement »

Il est bien joli de voir qu'un programme peut gagner facilement, mais comment doit-on s'y prendre en tant qu'être humain ? Je vais tenter d'expliquer comment utiliser la même méthode, mais d'une manière un peu différente de façon à ce que n'importe qui soit capable de l'utiliser.

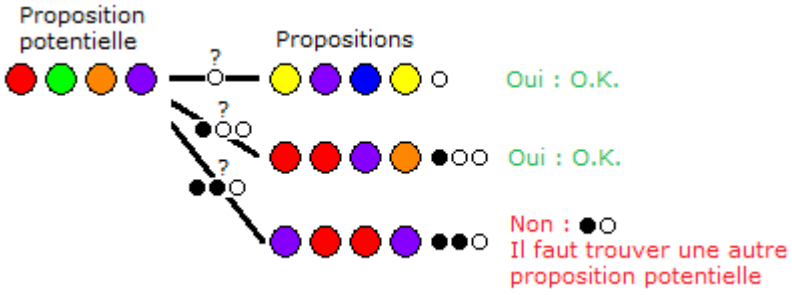
2.2.1. Explications

Il est évident que nous ne pouvons pas garder en mémoire un millier de combinaisons et les éliminer au fur et à mesure. Il faut donc utiliser une méthode dite conditionnelle.

On commence par proposer une combinaison aléatoire. Ensuite, à chaque coup, on choisit une combinaison de manière aléatoire ou assez intuitive par rapport aux résultats précédents. Avant de proposer cette combinaison, il faut qu'elle satisfasse un certain nombre de conditions : une par proposition précédente.

Comme il a été expliqué pour la méthode appliquée par l'ordinateur, il faut proposer une combinaison qui a le même nombre de pions noirs et de pions blancs avec les propositions précédentes que celles-ci en ont avec la solution.

A chaque coup, il faut vérifier les *X* conditions qui correspondent aux *X* coups précédents. Ainsi, au premier coup, on peut proposer ce que l'on veut. Notre proposition doit toujours avoir N_i noirs et B_i blancs en commun avec chaque proposition (d'indice *i*), N_i et B_i étant le nombre de noirs et de blancs que chaque proposition *i* a en commun avec la solution.



Pour éviter de perdre beaucoup de coups, il est impératif que ces conditions soient remplies. Autant dire que si vous faites une proposition qui ne remplit pas toutes ces conditions, elle ne valait pas la peine d'être proposée. Il faut persévérer. C'est comme résoudre un petit casse-tête à chaque coup qui devient de plus en plus limité en solutions, mais pas forcément plus difficile.

2.2.2. Exemple pas-à-pas

Puisque, j'en suis conscient, il peut-être difficile de comprendre la méthode par la théorie et les formulations un peu indigestes, je vais ici montrer un exemple détaillé d'une partie en utilisant cette méthode.

Je commence par choisir une combinaison totalement aléatoire. Je propose ceci et j'obtiens le résultat suivant :

Je dois maintenant proposer une combinaison qui a 1 noir avec celle-ci. J'ai évidemment l'embarras du choix. Je choisis de garder la dernière boule orange. Je ne dois donc pas mettre une autre boule orange, bleue ou jaune, sinon il y aurait conflit avec cette proposition (on obtiendrait autre chose qu'un unique pion noir). Je propose ceci :

On voit que ce n'était pas cette boule orange qui était bien placée. Peu importe. Je choisis une autre boule qui fera office de pion noir avec la première proposition. Je prends la troisième boule qui est bleue. Je ne dois pas mettre d'orange, de bleu ou de jaune (sinon il y aurait conflit avec l'unique noir de la première proposition) et pourtant je dois avoir 3 blancs avec la deuxième proposition. Je choisis de reprendre une violette, une rouge et une verte que je place différemment que dans la proposition 2.

J'ai maintenant toutes les bonnes boules. Il suffit de les mettre dans le bon ordre. Je garde ma boule bleue en troisième position qui servira a faire un pion noir avec la première et troisième proposition. Il faut maintenant placer la boule violette, la boule verte et la boule rouge.

La boule rouge ne peut être qu'en dernière position. En première position, il y aurait conflit avec la deuxième proposition. En deuxième position, il y aurait conflit avec la troisième proposition. En troisième position, j'ai déjà choisi de mettre ma boule bleue.

Même argument pour la boule violette, il ne reste que la deuxième position pour celle-ci. La verte se trouve donc dans le dernier emplacement possible.

On trouve donc la solution qui est :

4 coups, c'est plutôt bon.

2.2.3. Exercices

La théorie et les exemples c'est bien, mais c'est en pratiquant qu'on assimile le mieux les méthodes. C'est pour cela et aussi pour que vous puissiez vous amuser, ou plutôt réfléchir, que je vous mets ici des exercices dans lesquels il ne faudra pas faire une partie de Mastermind, ce qui serait impossible sans l'intervention d'une autre intelligence, mais dans lesquels il faudra trouver une combinaison qu'il faut proposer selon la méthode expliquée au-dessus en fonction des coups précédents déjà définis. Les exercices sont triés par ordre de difficulté croissante.

Trouvez une proposition « intelligente » (qui n'a aucun conflit avec les résultats précédents). Les solutions se trouvent juste après les exercices.

1)

2)

3)

2.2.4. Solutions

Voici les solutions pour les exercices. Elles ont toutes été calculées par ordinateur. Elles sont codées sous forme de série de chiffres pour une meilleure rédaction et aussi pour que vous ne puissiez pas tricher par inadvertance !
0 : Rouge ; 1 : Bleu ; 2 : Jaune ; 3 : Vert ; 4 : Orange ; 5 : Violet

1) [0, 3, 1, 4], [0, 4, 3, 1], [1, 0, 2, 4], [1, 0, 3, 2], [1, 3, 0, 4], [1, 3, 1,4], [1, 3, 3, 2], [1, 4, 2, 4], [1, 4, 3, 0], [1, 4, 3, 1], [3, 0, 1, 4], [3, 2, 0,1], [3, 2, 1, 0], [3, 2, 1, 1], [3, 2, 1, 2], [3, 2, 2, 1], [3, 3, 1, 4], [3, 4, 1,4], [3, 4, 3, 1], [4, 0, 3, 1], [4, 2, 0, 1], [4, 2, 1, 0], [4, 2, 1, 1], [4, 2, 1,2], [4, 2, 2, 1], [4, 3, 1, 4], [4, 3, 3, 1], [4, 4, 3, 1]

2) [0, 4, 1, 1], [1, 0, 4, 1], [1, 1, 4, 0], [1, 1, 4, 1], [2, 0, 1, 3],[2, 2, 1, 3], [3, 0, 3, 5], [3, 5, 3, 5]

3) [1, 1, 2, 2]

3. Mode d'emploi

3.1. Installer Python

Afin de pouvoir visionner et faire tourner mes programmes, il faut que Python soit installé sur votre ordinateur. S'il ne l'est pas, vous pouvez utiliser l'installateur que se trouve en annexe. Il s'agit de *python-2.6.4.msi*. Sinon, vous avez toujours la possibilité de le télécharger sur le site officiel : http://www.python.org.

Pour démarrer un programme, il suffit d'éditer le fichier *.py avec Idle et de cliquer sur Run (ou appuyer sur F5).

3.2. Le programme des statistiques

Il s'agit de *mastermind_stats.py* se trouvant sur le CD-ROM *Mastermind*

Comme il a été dit plus haut, ce programme sert à effectuer toutes les simulations pour obtenir des résultats afin de dresser des statistiques nécessaires à l'étude. C'est pourquoi ce programme se doit d'être simple, optimisé et rapide. Comme il n'est pas vraiment destiné à un utilisateur quelconque, il est dépourvu d'interface graphique et autres options superflues.

Il suffit de le démarrer et de remplir les champs. Le nombre de simulations correspond au nombre de parties qui seront simulées. Plus on en effectue, plus les statistiques seront précises. Pour ce qui est de la variété de couleurs, mettez toujours 0. Vous verrez plus loin de quoi il s'agit. En mettant 0, le programme générera toujours des solutions totalement aléatoires.

3.3. Le jeu

Il s'agit de *mastermind_jeu.py* se trouvant sur le CD-ROM *Mastermind*

C'est maintenant le moment de vous distraire. J'ai trouvé qu'il était incontournable de traiter le Mastermind en informatique sans faire une petite application pour y jouer et aussi pour vous montrer l'efficacité de ma méthode de manière plus reposante et interactive, car en effet, vous allez pouvoir jouer en étant, si vous le désirez, aidé par l'ordinateur.

Paramétrer une partie

Il faut commencer par choisir votre difficulté qui est exprimée par le nombre de boules, le nombre de couleurs et le nombre d'essais. Remplissez les champs prévus à cet effet. Je peux vous recommander une configuration « par défaut » qui est de 4 boules, 6 couleurs et 10 essais (bien que si vous maîtrisez parfaitement la méthode, vous passerez rarement au-delà de 6 coups). Une fois les valeurs entrées, vous n'avez qu'à cliquer sur « Jouer ».

ATTENTION :
Si vous tentez de jouer sur une machine peu puissante avec beaucoup de boules et de couleurs, il se peut que le programme se mette à utiliser toutes les ressources de l'ordinateur et reste bloqué pendant plusieurs secondes. Normalement, jouer avec 6 boules et 8 couleurs pourrait coincer le programme pendant environ 1 minute seulement après la validation de la première proposition. Les autres devraient être pratiquement instantanées.

Jouer

Comme vous le savez déjà, le but est de découvrir la combinaison cachée. Le tour que vous êtes en train de jouer correspond à la ligne portant l'indice entre crochets. Vous devez composer vos combinaisons en cliquant sur les boules de couleur se trouvant entre les lignes et les boutons. Vous verrez alors la ligne se remplir. Vous pouvez cliquer sur le bouton « Effacer » si vous pensez avoir fait une erreur, bien que si vous ajoutez des boules alors que la ligne est complète, elle sera automatiquement vidée et remplacée par la boule supplémentaire.

Pour valider votre proposition, vous n'avez qu'à cliquer sur le bouton « Proposer » comme vous ne vous en doutiez pas. Vous verrez alors le résultat apparaître sur la droite. Notez qu'à tout moment, vous pouvez cliquer sur le bouton « Nouvelle partie » pour recommencer.

Intéressons-nous au bouton « Vérifier ». Il vous permet, une fois votre combinaison composée, avant de l'avoir validée, de demander à l'ordinateur de contrôler si c'est une proposition « intelligente », c'est-à-dire qu'il n'y ait aucun conflit et qu'elle satisfasse bien toutes les conditions. Rappelez-vous, c'est ce que vous deviez faire à la partie « 2.2.3. Exercices ». C'est en fait un correcteur qui vous dira si votre proposition est recevable.

Quant à lui, le bouton « Conseiller » est carrément plus efficace. Ce que vous ne saviez peut-être pas, c'est que l'ordinateur utilise tous vos résultats et tente de trouver la solution avec vous (c'est pour cela que le programme consomme beaucoup de ressources lorsqu'on joue avec beaucoup de boules et de couleurs). En cliquant sur ce bouton, l'ordinateur vous donnera directement une proposition qui n'a aucun conflit. Ainsi, si vous êtes bloqué, il viendra directement à votre secours. D'une autre manière, si vous appuyez successivement sur les boutons « Conseiller » et « Proposer », vous verrez exactement comment fait l'ordinateur pour gagner en si peu de coups. Plus élégant que dans l'autre programme, n'est-ce pas ?

4. Analyse des résultats

4.1. Le nombre de coups pour gagner

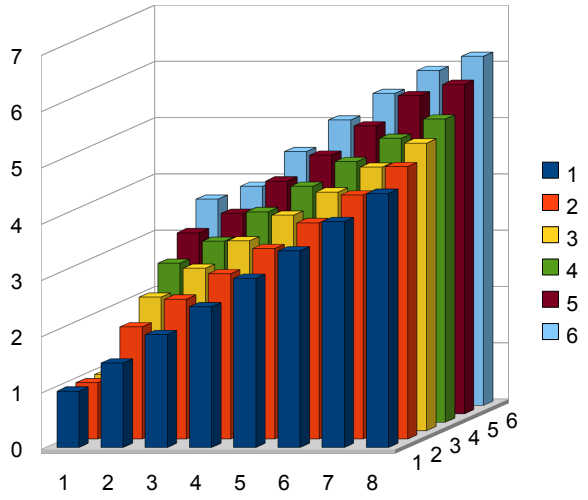
Grâce aux nombreuses simulations, j'ai pu découvrir le nombre de coups qu'il faut en moyenne pour découvrir la combinaison cachée en fonction du nombre de couleurs et du nombre de boules. J'ai effectué de 100 à 10'000 simulations par cas, cela dépend de la complexité de l'opération. Ainsi, plus le nombre de boules et le nombre de couleurs sont élevés, plus le nombre de simulations est bas, car sinon, cela prendrait trop de temps. Les valeurs sont donc moins précises pour ces cas.

Moyenne du nombre de coups en fonction du nombre de boules et de couleurs

	1 boule	2 boules	3 boules	4 boules	5 boules	6 boules
1 couleur	1.000	1.000	1.000	1.000	1.000	1.000
2 couleurs	1.503	1.996	2.373	2.822	3.216	3.663
3 couleurs	2.006	2.485	2.884	3.214	3.562	3.893
4 couleurs	2.499	2.938	3.371	3.740	4.135	4.507
5 couleurs	3.008	3.387	3.825	4.192	4.596	5.070
6 couleurs	3.495	3.844	4.238	4.631	5.117	5.540
7 couleurs	4.014	4.338	4.680	5.043	5.650	5.950
8 couleurs	4.513	4.841	5.109	5.388	5.850	6.200

Rouge : Mastermind classique

Graphique correspondant au tableau ci-dessus



axe Z (verticalement) : nombre de coups
axe Y (en profondeur) : nombre de boules
axe X (horizontalement de gauche à droite) : nombre de couleurs

On remarque qu'il n'y a pas de surprise pour la première ligne (1 couleur). Il y a beau avoir beaucoup de boules, s'il n'y a qu'une couleur, il n'y a qu'une seule solution. C'est pourquoi on trouve toujours en 1 coup. C'est évident.

En ce qui concerne la première colonne (1 boule), elle était aussi plutôt prévisible. En effet, puisqu'il n'y a qu'une seule boule, les indications ne nous servent à rien. Il s'agit simplement de chance. Soit on tombe sur la bonne couleur, soit non. On peut faire de 1 à n coups, n étant le nombre de couleurs. Cela est équiprobable de faire n'importe quel nombre de coups de 1 à n.

En effet, pour n couleurs, on a les probabilités suivantes :

$$P(1\,coup)=\frac{1}{n}$$

$$P(2\,coups)=\frac{n-1}{n}\cdot\frac{1}{n-1}=\frac{1}{n}$$

$$P(3\,coups)=\frac{n-1}{n}\cdot\frac{n-2}{n-1}\cdot\frac{1}{n-2}=\frac{1}{n}$$

...

Calculer la moyenne du nombre de coups revient alors à calculer l'espérance de gain de ces probabilités pondérées par leur nombre de coups.

$$M(n\,couleurs)=\sum_{k=1}^{n}\frac{1}{n}\cdot k$$

Évidemment, on n'obtient pas tout à fait ça dans le tableau, mais plus on fait de simulations, plus ça tendra vers cette valeur.

On remarque également qu'avec 6 boules et 4 couleurs, on arrive à une moyenne de coups plutôt basse : 4.631 coups. On est bien loin des classiques 12 essais autorisés au Mastermind.

4.2. Recherche de la combinaison la plus difficile à trouver

Après avoir calculé toutes les moyennes de coups nécessaires pour trouver la solution, il pourrait être intéressant de savoir quelles sont les combinaisons qui ont nécessité beaucoup de coups et celles qui ont été faciles à trouver.

C'est alors que j'ai modifié mon programme pour qu'il me donne, une fois toutes simulations effectuées, le nombre maximum de coups avec la ou les combinaison(s) associée(s). Cela n'a pas donné de résultat étant donné que le hasard a eu un rôle important. En effet, il peut arriver de temps en temps que l'ordinateur perde beaucoup de coups sur une combinaison « facile ». De plus, certains types de combinaisons ont plus de chances d'être générées que d'autres. Néanmoins, j'ai tout de même remarqué qu'aucune combinaison contenant des boules toutes de la même couleur n'a figuré dans la liste des combinaisons qui ont nécessité le plus de coups.

Exemple d'un rapport :
1000 simulations ont été effectuées.
Moyenne du nombre de coups : 4.647
Le plus grand nombre de coups est 7.
C'est arrivé 6 fois et c'était avec la/les solution(s) :
[[5, 4, 1, 4], [5, 4, 3, 2], [2, 3, 5, 2], [1, 5, 5, 3], [5, 4, 5, 5], [5, 0, 3, 4]]

Remarquez qu'ici, les couleurs sont codées par des chiffres, chaque chiffre correspondant à une couleur. Ainsi, l'ordinateur peut effectuer plus facilement certaines opérations et aussi parce qu'il serait inutile de travailler avec des chaînes comme « bleu », « rouge » etc...

La seule chose qu'on peut distinguer entre les combinaisons est le nombre de couleurs différentes utilisées. C'est ce que j'appelle la « variété de couleurs ». Il ne faut pas confondre la variété de couleurs avec le nombre de couleurs. Le nombre de couleurs concerne une partie entière : il s'agit du nombre de couleurs à disposition pour créer les combinaisons. La variété de couleurs concerne une combinaison et indique sa « complexité » à travers le nombre de couleurs différentes utilisées.

Exemple :
violet-vert-vert-bleu : complexité 3 (on a bleu, vert et violet)
rouge-rouge-jaune-jaune : complexité 2 (on a rouge et jaune)

La variété de couleurs peut aller de 1 au nombre de boules ou de couleurs en prenant celui des deux qui est le plus petit.

Exemple :
Pour 4 boules et 6 couleurs, la complexité maximum est 4. En effet, même si l'on dispose de 6 couleurs, on ne peut que mettre une couleur différente par boule, donc 4

Pour 6 boules et 4 couleurs, la complexité maximum est 4 aussi. Même si l'on a 6 boules qui pourraient être différentes, on n'a que 4 couleurs différentes.

J'ai ciblé mes mesures afin d'obtenir des résultats plus utiles. J'ai modifié ma fonction *creerComb* en lui ajoutant un paramètre supplémentaire qu'est la variété de couleurs. De cette manière je peux effectuer des simulations de parties avec une solution comportant un nombre défini de couleurs différentes. Pour chaque nombre de couleurs et de boules, je peux utiliser toutes les variétés de couleurs possibles et faire une moyenne. Notons que j'ai quand même gardé la possibilité de créer des combinaisons totalement aléatoires. J'ai effectué de 10'000 à 100 simulations pour obtenir ces résultats.
(c.f. *mastermind_stats.py* lignes 95-115)

Moyenne du nombre de coups en fonction du nombre de couleurs et de la variété de couleurs pour 4 boules

Nombre de couleurs	Variété de couleurs de la solution	Moyenne du nombre de coups
1	1	1.000
2	1	2.426
	2	3.690
3	1	2.759
	2	3.201
	3	3.255
4	1	3.257
	2	3.732
	3	3.783
	4	3.805
5	1	3.696
	2	4.167
	3	4.245
	4	4.334
6	1	4.098
	2	4.599
	3	4.652
	4	4.647*
7	1	4.459
	2	4.998
	3	5.102
	4	5.108
8	1	4.827
	2	5.387
	3	5.466
	4	5.458*

Rouge : Mastermind classique

* : Le hasard peut parfois briser la chaîne croissante du nombre de coups. En effet, plus le nombre de boules et de couleurs est grand, plus la différence de coups en fonction de la variété de couleurs s'estompe et plus le nombre de simulations diminue. Néanmoins, la différence entre variété 1 et plus reste très marquée.

On voit ici très clairement que plus la variété de couleurs est grande, plus il faut de coups pour trouver la solution. C'est-à-dire que plus la solution est faite de couleurs différentes, plus elle est difficile à trouver.

Le plus grand écart est entre variété 1 et plus. Il faut donc absolument éviter de faire une combinaison avec des boules toutes de la même couleur. Ce qui est assez intuitif. Il est carrément presque tout le temps plus difficile de trouver la solution avec une couleur de moins que d'avoir une solution unicolore. De plus, si l'on joue contre un joueur humain, il pourra se douter de la supercherie.

4.3. Nombre de possibilités éliminées en fonction des résultats

J'ai trouvé intéressant d'étudier le pourcentage de possibilités effacées en fonction du nombre de pions noirs et de pions blancs obtenus. Cela sert en quelque sorte à savoir quand l'on doit être « content » ou quand on peut s'estimer chanceux d'avoir obtenu un résultat.

J'ai donc utilisé une version modifiée de *mastermind_stats* pour qu'il me calcule ce pourcentage de possibilités éliminées au premier coup. J'ai ainsi pu dresser le tableau suivant avec 4 boules et 6 couleurs à l'aide de 10'000 simulations par cas.

Pourcentage de possibilités éliminées au premier coup avec 4 boules et 6 couleurs

	0 noir	1 noir	2 noirs	3 noirs	4 noirs
0 blanc	92.872%	86.163%	91.854%	98.457	99.923%
1 blanc	81.013%	82.549%	96.934%	-*	-
2 blancs	82.781%	93.228%	99.615	-	-
3 blancs	95.206%	99.648%	-	-	-
4 blancs	99.709%	-	-	-	-

*Remarquez qu'on ne peut avoir 3 pions noirs et 1 pion blanc. En effet, si tous les pions sauf 1 sont bien placés, le dernier ne peut être qu'à sa place.

On remarque, sans grand étonnement, que plus le nombre de pions noirs augmente, plus le pourcentage de possibilités éliminées est grand, tout comme pour les pions blancs. Mais en revanche, à ma grand surprise, il y a un cas auquel je n'avais pas pensé. Obtenir un résultat de 0 blancs et 0 noirs permet d'éliminer bien plus de solutions que je l'imaginais. À un tel point, qu'il est préférable d'obtenir 0 blancs et 0 noirs plutôt que simplement 2 noirs.

On remarque également, que d'une manière générale, tous les pourcentages sont assez hauts : pas un cas en dessous de 80 %. On ne peut donc pas parler « d'échec critique ». Le procédé permet d'éliminer des possibilités de manière assez linéaire.

Notons également que le résultat 4 noirs et 0 blanc n'est autre que $\frac{1295}{1296}$, ce qui est le nombre de possibilités moins un sur le nombre de possibilités. En effet, avec 4 pions noirs, on peut tout éliminer sauf notre proposition qui est la solution.

5. Récapitulatif

C'est alors qu'après toutes ces analyses, on peut dire les choses suivantes :
il y existe une méthode plutôt simple, qui correctement exécutée, permet de gagner facilement une partie de Mastermind. Si l'on joue sur une table de jeu normale, on peut être facilement capable de venir à bout de la combinaison cachée (on pourrait carrément y arriver 2 fois). Néanmoins, on se rend compte que le jeu devient très vite compliqué lorsqu'on augmente le nombre de boules et de couleurs : un ordinateur a déjà extrêmement beaucoup de peine avec 8 boules et 8 couleurs, avec mon programme bien entendu. Peut-être qu'il y existe un moyen plus efficace d'y arriver ?
On se rend compte que la combinaison la plus difficile à trouver est celle qui contient le plus de couleurs différentes, alors que la plus facile à trouver est l'unicolore. On remarque aussi que le procédé de déduction de la solution est plutôt linéaire et n'est pas instable du tout : on n'est pas forcément plus avancé en ayant obtenu un pion noir au premier coup, même si plus ceux-ci sont nombreux, plus on s'approche de la solution.
Le Mastermind est un jeu qui en a probablement effrayé plus d'un et qui pourtant n'est pas si compliqué que cela.

6. Conclusion

Arrivé au terme de mon travail de maturité, j'espère que vous avez pris du plaisir à me lire et à essayer mes programmes. J'ai trouvé cette expérience très enrichissante, car ça m'a permis d'utiliser mes connaissances acquises l'année précédente en informatique pour étudier un sujet plutôt ludique. J'espère que, tout comme moi, vous avez progressé dans ce jeu qui m'a toujours paru plus compliqué qu'il ne l'était et que vous ne vous laisserez plus jamais battre. Je suis aussi plutôt content, car j'estime que j'ai tout de même bien réussi ce que j'ai entrepris : les programmes fonctionnent et les statistiques étaient intéressantes.

7. Informations sur la réalisation

Toutes les mesures ont été effectuées sur un dual core AMD 3.1Ghz avec 2.0Go de RAM. Le temps total de mesures s'est élevé à 4 heures et 42 minutes, ce n'était peut-être pas nécessaire, mais je souhaitais obtenir des valeurs précises. J'ai d'ailleurs dû laisser tourner mon ordinateur plusieurs fois en mon absence. Le code présent dans mes programmes a été entièrement écrit par moi-même.

Sites internet consultés :
http://fr.wikipedia.org/wiki/Mastermind
http://www.apprendre-en-ligne.net/python
http://www.python.org

Déclaration

Je déclare par la présente que j'ai réalisé ce travail de manière autonome et que je n'ai utilisé aucun autre moyen que ceux indiqués dans le texte. Tous les passages inspirés ou cités d'autres auteur-es sont dûment mentionnés comme tels. Je suis conscient que de fausses déclarations peuvent conduire le Lycée cantonal à déclarer le travail non recevable et m'exclure de ce fait à la session d'examens à laquelle je suis inscrit.
Ce travail reflète mes opinions et n'engage que moi-même, non pas le professeur responsable de mon travail ou l'expert qui m'a accompagné dans cette recherche.

Lieu et date : ..... Signature : .....