Mastermind

Mes règles :

Inspiré du traditionnel mastermind, mes règles sont simples. Avant tout, j'utiliserai les couleurs suivantes : gris, blanc, rose, orange, vert, bleu, rouge, jaune. Le jeu débute de la façon suivante : je crée un code secret de 4 couleurs. Mon but est de deviner en un minimum de coups cette combinaison. Je gagne une manche dès que j'y parviens en maximum 2 coups pour un niveau expert, 12 coups pour un niveau intermédiaire et 18 coups pour un niveau débutant. Une fois le code secret établi, je commence à faire des propositions de combinaisons en insérant 4 couleurs (pas forcément différentes) dans le code. Si ma combinaison présentée est correcte, je reçois un message me faisant part de ma victoire. Si la combinaison est incorrecte, je donne certaines indications à mon rival : d'une part le nombre de bonnes couleurs, c'est-à-dire les couleurs présentes à la fois dans le code secret et dans ma proposition (bien et mal placées confondues), et d'autre part le nombre de couleurs bien placées cette fois-ci.

Exemple : code secret : [Vert, Rose, Blanc, Gris]
Proposition 1 : [Gris, Gris, Bleu, Rouge]
- ⇨ 1 bonne couleur
- ⇨ 0 couleur bien placée

Proposition 2 : [Vert, Gris, Rose, Gris]
- ⇨ 3 bonnes couleurs
- ⇨ 1 couleur bien placé

parties() : Procédure qui demande au joueur de choisir entre la partie1 et la partie2 :
- Partie 1 : ordinateur fixe code secret => joueur chercher le code
- Partie 2 : joueur fixe code secret => ordinateur cherche le code

partie1() : Procédure qui lance la partie 1, suite au choix du joueur

partie2() : Procédure qui lance la partie 2, suite au choix du joueur

Partie 1 :

La première partie du code vous permet de jouer contre l'ordinateur. Il génère un code secret aléatoirement et c'est à vous de le trouver.

code_secret( ) : Première fonction du code, l'ordinateur génère un code secret munit de 4 éléments piochés aléatoirement dans la liste de couleur suivante : ["gris","blanc","rose","orange","vert","bleu","rouge","jaune"].

menu( ) : Procédure affichant les différentes couleurs disponibles pour que le joueur forme ses propositions.

traduction_menu(x) : Le joueur entrant une matrice de 4 chiffres correspondant à des couleurs choisit, cette fonction vient traduire ces chiffres en couleurs.

code_secret_professeur( ) : Fonction dédiée à un mode particulier, le professeur entre sa propre combinaison secrète et joue contre lui-même.

code_joueur( ) : Fonction qui renvoie la proposition du joueur avec les chiffres traduit via la fonction de traduction cité plus haut.

comparaison(combinaison,proposition) : Fonction qui renvoie après avoir balayé chaque éléments des deux combinaisons (celle de l'ordinateur et celle du joueur), si il y en a, le nombre

de couleur que les deux listes ont en commun et si il y en a, le nombre de couleur bien placées au même endroit dans les listes.

****** On affecte à la variable coul_pos la valeur de retour de la fonction de comparaison, à savoir une matrice de deux réels tel que le premier élément de la matrice corresponde au nombre de couleur en commun entre les 2 combinaisons et le deuxième élément de cette matrice soit le nombre d'éléments bien positionné ******

resultat(coul_pos) : procédure qui affiche le nombre de couleurs correctes et les nombres d'éléments bien placées.

rappel(memoire.proposition) : Cette procédure affiche la mémoire des propositions du joueur, comme dans le vrai jeu finalement, il les a toutes sous les yeux.

test_combinaison_correcte1(position, combinaison , proposition, memoire) : Procedure qui tant que la variable position est differente de 4, i.e. le joueur n'a pas la combinaison correcte, rejoue : redemande une composition au joueur + compare cette nouvelle combinaison à celle proposée par l'ordinateur. Destinée au mastermind_tricheur() car le joueur a accès au code dit secret fixé par l'ordinateur.

test_combinaison_correcte2_debutant(position, combinaison , proposition, memoire) : Exactement le même rôle que la précédente. Cette fois-ci destinée au mastermind() classique, le code secret n'est pas visible. Plusieurs niveau de jeu sont possible : DEBUTANT, INTERMEDIAIRE, EXPERT. Comme son nom l'indique ici nous nous trouvons dans le niveau DEBUTANT : 18 coups possible pour trouver le code secret et remporter la partie.

test_combinaison_correcte2_intermediaire(position, combinaison , proposition, memoire) : La même chose qu'au dessus mais dans le niveau INTERMEDIAIRE : plus que 12 coups possible pour trouver le code secret et remporter la partie
test_combinaison_correcte2_expert(position, combinaison , proposition, memoire) : Pas de surprise, pareil que les deux précédentes en niveau EXPERT : seulement 2 coups possible pour trouver le code secret et remporter la partie .

test_combinaison_correcte3(position, combinaison, proposition, memoire) : Procédure qui tant que la variable position est différente de 4, i.e. le joueur n'a pas la combinaison correcte, rejoue : redemande une composition au joueur + compare cette nouvelle combinaison à celle proposée par le professeur. Destinée au mastermind_special_professeur().

mastermind_tricheur() : Le jeu version mauvais joueur, vous avez accès au code secret choisit par l'ordinateur. Cette procédure lance le jeu à l'aide des fonctions créer dans le code (voir démonstration).

mastermind_debutant()/ mastermind_intermediaire()/ mastermind_expert() : Le jeu, le vrai! Nous avons pensé à tout le monde, à tous les niveaux : DEBUTANT, INTERMEDIAIRE et EXPERT.

mastermind_special_professeur() : Le jeu version professeur, c'est-à-dire celle où l'utilisateur entre son propore code secret et joue contre lui-même.

menu_mode_de_jeu() : Procédure proposant au joueur les différents modes de jeu coder ci-dessus : « tricheur », « classique », « professeur ».

menu_niveau() : Procédure indiquant les niveaux de jeu i.e. EXPERT, INTERMEDIAIRE, DEBUTANT, la différence réside dans le nombre de coups limitant la partie, je contraints le niveau expert à 2 coups pour montrer ce qu'il se passe : le programme s'arrête bien.

niveau(niv) : Fonction qui en fonction de l'indication de l'utilisateur sur son niveau de jeu lance un mode de jeu plus ou moins compliqué c'est à dire le nombre de coups est +/- limité. Cette fonction est appelée seulement si le joueur joue au mastermind classique, elle est inutile dans les autres versions du jeu.

mode_de_jeu(choix) : Suite à l'affichage des modes de jeu, l'utilisateur entre son choix, on récupère ce choix dans cette procédure qui va lancer le mode de jeu correspondant.

Partie 2 : Les heuristiques, nous inversons les rôles

Cette fois-ci le joueur détermine un code secret composé de 4 couleurs (pas forcement différentes). L'ordinateur va tenter de déterminer la combinaison choisit par l'utilisateur, pour cela il a le choix entre plusieurs stratégies +/- efficaces.

code_secret_utilisateur() : Cette fonction demande au joueur de saisir 4 nombres et retourne une liste de couleur représentant le code secret.

erreur(couleurs): Fonction qui vérifie que les entiers renseignés sont compris entre 1 et 8, gestion d'erreur.

combinaison_alea() : La fonction combinaison retourne une combinaison de 4 couleurs proposée par l'ordinateur. La proposition est générée aléatoirement parmi la liste des 8 couleurs du jeu ([Gris, Blanc, Rose, Orange, Vert, Bleu, Rouge]).

couleurs_secretes() : Cette fonction va gérer une mémoire qui vient stocker les couleurs présentent dans le code secret. Pour trouver ces couleurs l'ordinateur va proposer 8 combinaisons, chacune d'elle sera composé de 4 couleurs identique. Une fois les couleurs présentent dans le code secret trouvée, on vient les placer dans une matrice que cette fonction nous retourne.

Exemple : code secret : [Vert, Rose, Blanc, Gris]
Proposition 1 : [Gris, Gris, Gris, Gris]
- ⇨ 1 bien placé
- ⇨ Matrice = [Gris]

Proposition 2 : [Rose, Rose, Rose, Rose]
- ⇨ 1 bien placé
- ⇨ Matrice = [Gris, Rose]

Etc…

couleurs_secretes_efficaces(combinaison): Cette fonction reprend le même principe que couleurs_secretes mais s'arrête dès que la mémoire contient 4 couleurs. Comme le nombre de coups varie pour cette fonction, elle retourne aussi le nombre de coups utilisés

combinaison_couleurs_reduites(mémoire) : Cette fonction renvoie des propositions tirées aléatoirement par l'ordinateur mais cette fois-ci les couleurs sont piochées dans la liste "memoire" qui a stockée les couleurs présentent dans le code secret, cela réduit considérablement le nombre de combinaison possible.

comparaison_part2(secret,proposition): Comme son nom l'indique cette fonction va comparer le code secret et la proposition de l'ordinateur elle retourne ensuite le nombre de couleurs bien placées.

position(couleur,secret): Procédure qui détermine la position des couleurs retenus i.e. celle présentent dans le code secret. Pour cela elle utilise une couleur inexistante (violet). Utilisé dans l'heuristique « efficace ».

Exemple : code secret : [Vert, Rose, Blanc, Gris]
Couleurs secrètes retenu de la fonction « couleurs_secretes() » : [Gris, Blanc, Vert, Rose]
Proposition 1 : [Gris, Violet, Violet, Violet]
⇨ Trouvé = False
Proposition 2 : [Violet, Gris, Violet, Violet]
⇨ Trouvé = False
Proposition 3 : [Violet, Violet, Gris, Violet]
⇨ Trouvé = False
Par défaut : dechiffre=[0,0,0, Gris] et indices_choisis=[3]

Proposition 1 : [Blanc, Violet, Violet, Violet]
⇨ Trouvé = False
Proposition 2 : [Violet, Blanc, Violet, Violet]
⇨ Trouvé = False
Par défaut : dechiffre=[0,0, Blanc, Gris]et indices_choisis=[3,2]

test_combinaison_correcte_alea(secret,cbp): Cette procédure invite l'ordinateur à rejouer une proposition tant qu'il ne trouve pas le code secret, une fois ce dernier trouver elle nous renvoi le nombre de coups qu'il lui aura fallu. Destiné à l'heuristique « aleatoire ».

test_combinaison_correcte_pas_completement_alea(secret,cbp,couleur) : Cette procédure est identique à celle au-dessus à la différence près que l'ordi pioche aléatoirement mais dans une liste de maximum 4 couleurs (en effet il peut y avoir plusieurs couleurs identique dans le code secret). Destiné à l'heuristique « pas_completement_aleatoire ».

menu_heuristique(): Procédure qui affiche les différentes heuristiques (3 au total).

mode_de_jeu_part2(choix): Procédure qui lance l'heuristique choisit par l'utilisateur.

Heuristique 1 : Aléatoire : Dans cette première stratégie, l'ordinateur va tirer aléatoirement des combinaisons tant qu'il n'a pas trouver la combinaison secrète.
aleatoire()

Heuristique 2 : Pas complètement aléatoire : l'ordinateur détermine dans un premier temps les couleurs présentent dans le code secret suite à cela, il tire aléatoirement des combinaisons seulement composé des couleurs secrètes.
pas_completement_aleatoire()

Heuristique 3 : Efficace : Elle reprend comme ci-dessus les couleurs présentent dans le code secret puis les comparent une à une au code secret, une fois la position de la couleur trouvée on remplit une matrice avec les couleurs positionnés au bon endroit.
efficace()

Conclusion :

Mes heuristiques évoluent de manière croissante. Ma première heuristique n'est pas très efficace, nécessitant un nombre élevé de coups en moyenne (4096 coups). Le deuxième réduit considérablement le nombre de coups (en moyenne 206), tandis que la troisième et dernière limite à un maximum de 14 coups. Cela me semble très efficace, surtout étant donné que dans le jeu de plateau classique, le nombre de coups est limité à 12. En conclusion, j'ai trouvé ce projet très enrichissant. Il m'a poussé à m'écouter, à m'adapter aux autres, car un code n'a pas une unique façon d'être programmé. Il a fallu discuter, chercher des solutions et les optimiser. Ce projet m'a beaucoup formé, et j'ai pu constater une évolution de mon niveau. Le sujet était intéressant, sérieux, mais aussi agréable, évoquant des souvenirs de jeux en famille.

Un grand merci à Mr. Berro de m'avoir soutenu et aidé tout au long de ce projet et de ce semestre.