****Algorithmie - Programmation 2 – TD008

Professeur : Kiouche Abderrahmane

**DOSSIER D’ANALYSE – PROGRAMMATION**

**MIASHS 1ere Année**

**Jeu : Mastermind**

Nathan Falcoz-Vigne (2218618) et Hubert Geoffray (2215445)

Le 2 mai 2022

Table des matières

[1) Revue de notre travail 3](#_Toc102433167)

[2) Annexe 1 : Dictionnaire de variable 5](#_Toc102433168)

[3) Annexe 2 : Algorithme 6](#_Toc102433169)

[4) Conclusions personnelles 8](#_Toc102433170)

1. Revue de notre travail

L’objectif du projet était de faire un jeu avec l’ambition de modéliser la stratégie d’un joueur « artificiel » ou une IA avec au moins deux adversaires. Le jeu Mastermind était proposé alors nous l’avons pris. L’enjeu du projet était de coder une IA capable de jouer et de réfléchir comme un humain qui pourrait jouer contre nous. Notre problématique est la suivante :

Comment programmer une IA capable de réfléchir comme un humain au jeu du Mastermind (notamment quand l’IA doit trouver notre code) ?

Le jeu du Mastermind ayant deux parties distinctes (codeur ou décodeur), nous avons choisi de nous partager chacun une partie du code et de travailler chacun de notre côté avant de combiner le code en un seul. Nathan s’est occupé de la partie codeur et Hubert de la partie décodeur mais nous avons quand même réfléchit sur comment nous allions coder pour chaque partie. C’est pour cela qu’au niveau de la « syntaxe » du code selon la partie n’est pas la même. Par ailleurs, nous avons préféré travailler sur des codes avec des chiffres au lieu des couleurs.

Limite : Qu’avons-nous laissé de côté ? Les doublons ne sont pas autorisés. En effet, dans les règles du jeu, les codes avec des couleurs ou chiffres peuvent se répétés. Or, on a commencé à coder avec la possibilité que les chiffres se répètent mais cela rendait notre tâche vraiment complexe. C’est pour cela qu’on a préféré simplifier les règles. Quand l’IA doit trouver le code on lui indique si un chiffre est bien positionné au lieu de lui dire le nombre de chiffre correct et bien positionné. Le fait de dire le nombre de chiffre correct et bien positionné est aussi handicapant pour nous. Alors on a simplifié pour l’IA seulement pour la partie codeur. Mais on a laissé cette règle pour la partie décodeur ou l’IA n'a pas de difficulté pour dire le nombre de chiffre bien placé et juste.

Structure de données : Nathan a préféré utiliser deux tableaux : un tableau « TM » pour stocker les valeurs du code et un tableau « tab2 » pour stocker les valeurs choisit par l’IA. Hubert à préféré utiliser un tableau pour le code de l’IA et stocker les valeurs du joueur. Dans les deux cas, les tableaux (qui font offices de plateau de jeu comme le jeu en physique) qui affichent le jeu sont tout le temps affichés.

Explication des choix effectués : Partie codeur : L’IA choisit des valeurs aléatoires parmi les valeurs possible (0-5) en évitant les doublons. On indique à l’IA si une valeur est correct et si elle est à la bonne position une par une, à partir de là l’IA va stocker des valeurs qu’elle sait ne pas être contenu dans le code, et elle va stocker les valeurs dont elle connaît la position, ce jusqu’à trouver le code choisit. Pour la partie décodeur, l’IA choisit son code aléatoirement entre 0 et 5. Pour chaque code entré par le joueur, l’IA indique le nombre de chiffre juste et juste et bien placé. La stratégie pour le décodeur (IA et le joueur) est d’essayer d’avoir dans un premier temps tous les chiffres « justes » et ensuite pouvoir les ordonner pour trouver le bon code.

Une éventuelle nouveauté Lazarus utilisée est la librairie « crt » (uses crt ;) qui nous permet d’utiliser le clear screen (clrscr ;) qui fait un affichage plus « propre » car il nettoie les données sorties avant le clear screen et affiche ce qu’il y après.

Mode d’emploi : En tant que codeur vous devrez choisir un code à quatre chiffres entre 0 et 5 et sans doublons. L’IA vous demandera d’indiquer pour chaque valeur si elle est correcte et à la bonne position. Le décodeur aura un total de 10 essais pour trouver le code choisit par l’IA aléatoirement. Pour jouer, il suffit de cliquer sur « Mastermind.exe » et d’autres explications seront dedans.

Au fur et à mesure de notre programmation, on lançait des tests pour régulièrement pour voir si tout fonctionnait normalement. On n’a pas trouvé la nécessité de créer un autre fichier Lazarus pour tester la validité de notre jeu. Parmi toutes les fonctionnalités que nous avons choisi d’intégrer, tout semble fonctionner correctement.

Pour conclure, le programme pourrait être amélioré en ajoutant la possibilité d’avoir des doublons. Pour la partie codeur dire seulement le nombre de chiffre juste et le nombre de chiffre bien placé, comme dans les règles du jeu de base. Rajouter des statistiques pour le joueur : nom/pseudo, nombre de victoire/défaite, ratio victoire/défaite moyenne de coup pour gagner (sur une session de jeu). Possibilité de rajouter plusieurs niveaux notamment sur le nombre de chiffres total. Dans nos règles c’est des 6 chiffres (0, 1, 2, 3, 4, 5), on aurait pu faire un niveau intermédiaire avec 8 chiffres et un autre avec 10 chiffres.

1. Annexe 1 : Dictionnaire de variable

|  |  |  |  |
| --- | --- | --- | --- |
| Variable | Type | Entrée/Interne/Sortie | Explication |
| Rep | Entier | Entrée | L’IA demande si la valeur est correcte rôle entier entré choisir le rôle codeur ou décodeur |
| Case1… Case4 | Entier | Interne | L’IA qui met les valeurs dans les différentes cases |
| Ban1 | Entier | Interne | Une valeur dont l’IA est sûre qu’elle n’est pas dans le code |
| Ban2 | Entier | Interne | Une valeur dont l’IA est sûre qu’elle n’est pas dans le code |
| I, j, j1, x1, y1 | Entier | Interne | Utilisés pour la construction des tableaux |
| Decodeur1 | Entier | Interne | Permet de faire fonctionner la partie décodeur dans la grande boucle while |
| Decodeur2 | Entier | Entrée | Permet de refaire une partie |
| Jouer | Entier | Interne | Permet de faire fonctionner le jeu en boucle tant que le joueur n’arrête pas |
| Rôle | Entier | Entrée | Permet au joueur de choisir son rôle |
| Nbre\_coup | Entier | Sortie | Permet de savoir en combien de coups l’IA a trouvé le code |
| Refaire | Entier | Entrée | Permet de refaire une partie après avoir perdu ou gagné dans la partie codeur |
| Win | Entier | Interne | Permet de savoir si l’utilisateur a gagné |
| Juste1, … juste4 | Entier | Interne | L’IA qui compte le nombre de chiffres justes |
| Tot\_juste | Entier | Interne | Somme du nombre des chiffres justes |
| Code1, … Code4 | Entier | Interne | Les codes correspondent au code que l’IA va choisir au début |
| Double1, … double3 | Entier | Interne | Cela évite les doublons de l’IA |
| Bon | Entier | Interne | Nombre de chiffre juste et bien placé |
| Choix | Entier | Entrée | Choix des codes que le joueur va mettre |

1. Annexe 2 : Algorithme

Début

Initialisation des variables

Ecrire(MASTERMIND)

Ecrire(REGLES DU JEU)

Demander(role)

Si (Role = 1) Alors // (Partie codeur)

Demander(Code)

Tab2[1,1] -> random(6)

Tab2[2,1] -> random(6)

Tant que (Tab2[1,2] = Tab[1,1]) Faire

Tab2[1,2] ->random(6)

Fin Tant que // On traite les potentiels doublons et on random pour le reste

Ecrire(tab2) // Affichage tableau

Demander(rep) // Si 1ere valeur est comprise dans le code

Si (Rep = 1) Alors

Tab2[5,1] -> tab2[5,1] + 1 // Nombre de chiffres corrects

Demander(rep) // SI cette valeur est a la bonne position

Si (rep = 1) Alors

Tab2[6,1] -> tab2[6,1] + 1 // Nombre de chiffre corrects et bien placés

Case1 -> tab2[1,1] // L’IA garde ce chiffre dans la case

Sinon

Ban1-> tab2[1,1] // Si le chiffre n’est pas bon

Fin Si

Fin si

// Après c’est des tests si la case n’est pas trouvée alors l’IA random

Si (code = code de l’IA) Alors

Ecrire(‘Partie terminée l’IA a trouvé le code’)

Fin Si // Et répéter cela 10 fois

Fin Si

Si (role = 2) alors // Partie décodeur

Pour j -> 1 à j-> 4 Faire

TM2[1,j] -> random (6) // L’IA qui choisit son code

Fin Pour

Tant que (TM2[1,1] = TM2[1,2]) Faire // Traitement des doublons possibles

TM2[1,2] -> random (6)

Fin Tant que // Même chose pour le reste

Demander(choix) //Entrée code du joueur

Pour j -> 1 à j -> 4 Faire

Si (TM2[1,j] = TM2[11,j] Alors

Bon -> bon + 1 // Nombre de chiffre juste et bien placé

Fin Si

Pour J -> 1 à j -> 4 Faire

Si (TM2[1,1] = TM2[11, j] Alors

Juste -> juste + 1 // Nombre de chiffre juste

Fin Si // Faire cela pour le reste

Ecrire (TM2[i,j]) // Affichage tableau

Si (win = 1) alors

Ecrire (Vous avez deviné le code de l’IA)

Fin Si // Et répéter cela 10 fois

Fin Si

1. Conclusions personnelles

**Nathan :**

J'ai beaucoup pris de plaisir à utiliser Lazarus, sa proximité à un algorithme pure m'a permis de mieux comprendre le fonctionnement d'un programme et de plus m'a aidé à réaliser l'importance d'un algorithme.

**Hubert :**

Ayant déjà fait un peu de programmation sur Python en classe de 1ere, cela m’a beaucoup facilité pour apprendre les bases de la programmation. Lazarus est très semblable à un schéma algorithmique ce qui me plait bien. Cet enseignement m’a confirmé que la programmation me plaisait plutôt bien et j’aimerai par la suite continuer dans ce domaine là en restant en Miashs.