**Rapport – Saé Dev n°2**

**Léo Henriot :**

Pour commencer, j’ai pris beaucoup de temps avant de me pencher sérieusement sur le sujet. J’ai commencé par proposer quelques idées à Gabriel (qui n’ont pas réellement été utiles (c’est inexacte, nous avons eu les mêmes idées donc elles étaient déjà implémentées -Gabriel)), et plus tard j’ai commencé à coder une IA pour les ronds. Mon idée est la suivante :

Fonctionnement

* On vérifie s’il y a déjà des ronds. S’il n’y en a pas, on le place dans un endroit propice aléatoirement, c’est-à-dire à au moins 4 cases de tous les bords.
* Autrement on procède de cette manière :

L’IA parcours tout le tableau :

* Si la case est un espace, on analyse dans chaque direction pour compter :
* Le nombre de ronds consécutifs dans cette direction
* Le nombre d’espace ainsi que de ronds dans la direction opposée (Afin de déterminer si l’on a suffisamment de place pour se développer dans cette direction)
* Si l’IA comprend que cette place est suffisante, alors elle stock les coordonnées dans 2 variables.
* Elle stock aussi la longueur des ronds déjà présent dans une variable,
* Si l’IA tombe sur un alignement de ronds supérieur, alors il priorise celui-ci et réenregistre les valeurs,
* Pour terminer, l’IA place le pion dans les coordonnées qui lui convient le mieux (C’est-à-dire là où il y a le plus de ronds ainsi qu’un espace suffisant).

Difficultés rencontrées

J’ai rencontré quelques difficultés m’ayant fait perdre du temps, mais en y réfléchissant c’étaient principalement des erreurs d’inattention.

* Premièrement je me suis retrouvé avec des segmentation fault qui se réglaient simplement avec une condition dans un while qui détecte que l’on ne sorte pas du cadre lors d’une recherche dans les directions.
* Ensuite je me suis rendu compte que j’avais oublié le principal, une condition if qui vérifie s’il y a un rond O, car le programme faisait juste compter les cases dans toute la largeur.
* Et 3ème problème important que j’ai rencontré, et celui-ci m’a malheureusement fait perdre la plupart du temps sur ce projet, je faisais un parcours du tableau toujours en colonne, c’est-à-dire que mes compteurs pos\_x et pos\_y étaient tout les deux dans tab[ici][j]… J’avais oublié de remplacer le j en gros. Après cela tout est rentré dans l’ordre et j’ai pu terminer correctement.

Modifications

Je me suis aussi permis de modifier le morpion.cpp ainsi qu’ajouter une phrase lorsque les ronds l’emportent afin d’avoir un rendu visuel plus facile (Les ronds en bleus et les croix en rouge).

Résultat

Exemple de partie contre cette IA (Ronds - IA, Croix - Moi-même) :



Je laisse maintenant Léo continuer. (Et pas moi-même haha… C’était pas drôle (je confirme - Gabriel))

**Léo Busch :**

**Gabriel OGER :**

J’ai commencé par chercher (sur internet) une idée d’IA de laquelle je pouvais m’inspirer, sans réel succès j’ai donc pris un papier et un crayon (il n’y a rien de mieux) et commencer à réfléchir de par moi-même, avec Léo H nous avons eu l’idée de baser la recherche sur un système de score représentant le danger, c’est avec l’autre Léo que nous avons trouvé la première étape : rechercher dans une plus petite grille. Après bien trop longtemps, j’ai enfin trouvé un moyen de calculer le danger : case par case, on applique une suite de règles. Bien entendus, ces règles ont changé avec le temps, elles ne sont plus les mêmes aujourd’hui. Après avoir créé un algorithme POC (Proof Of Concept) j’ai passé mon temps à le corriger et à le rendre plus lisible. En bref, cet algorithme applique des règles à chaque case dans un rayon donné, d’où son nom « BRBARA » (BRBARA is a Rule Based Area Restrained Algorithm).

Fonctionnement

* On commence par trouver un rond
* On définit une zone de recherche, ici on utilise 5 comme c’est le nombre de points à aligner pour gagner
* Pour chaque case dans cette zone, on recherche les joueurs dans plusieurs directions :
  + En Haut
  + En Bas
  + A gauche
  + A Droite
  + Chaque moitié des deux diagonales
* Dans ces direction on applique ces règles (*ia/croix/rules.cpp :20*) :
  + Pour chaque joueur O trouvé, on incrémente le danger de la case courante et on fait de même avec le nombre de O consécutifs
  + Si l’on trouve un joueur X alors on arrête de chercher puisque l’alignement n’est plus possible, de plus on efface le nombre de O consécutifs parce que ceux-ci sont séparés d’une croix
  + Enfin, si on trouve un espace, on efface le nombre de O consécutifs puisque ceux-ci sont séparés d’un espace
* Une fois le score de la case calculé, on recommence tout et on pose le pion sur la case avec le plus haut danger.

Difficultés rencontrées

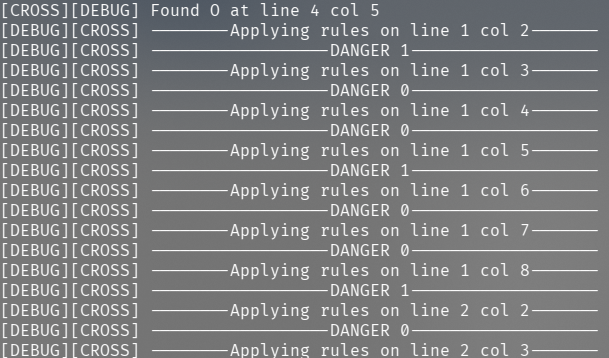
J’ai eu plusieurs problèmes :

* En premier lieu je me suis perdu à plusieurs reprise dans les coordonnées :/
* J’ai eu affaire à beaucoup de problème avec ma sous-grille et mon système de décalage, causant pas mal de pions mal placés ou un score incorrecte.
* Le problème qui est survenu le plus souvent est un accès illégal à la mémoire, ce qui était résolue assez rapidement à l’aide de GDB et des corefiles, cependant leur nombre d’occurrences reste assez élevé et j’ai donc perdu pas mal de temps.

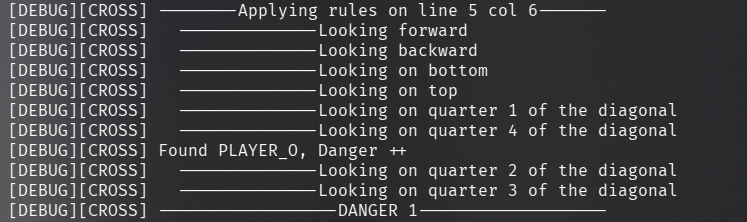
Outils supplémentaires

A des fins de débugging intense j’ai implémenté plusieurs outils :

* L’affiche du Danger par case (activable en compilant avec **-DDebugLvlLight**)



* L’affichage de l’application des règles (activable en compilant avec **-DDebugLvlMedium**)



* Une Heatmap pour afficher le score (activable en compilant avec **-DShowHeatmap**)



Ces outils sont très pratiques puisqu’ils montrent que plusieurs règles ne marchent pas :

* Diagonales
* O consecutifs

**Romain:**

La fonction « void jouerIA » de mon algo permet à une IA de jouer.

Dans l’algo il y a en premier 2 boucles « for » qui permettent de parcourir les positions du tableau. Ensuite on vérifie si l’emplacement est vide grâce à la condition « estLibre\_morpion » en vérifiant s’il n’y a pas déjà un rond ou une croix.

Le « placer\_morpion » permet de placer une croix ou un rond ce qui après avoir vérifié l’emplacement. A la suite de cela l’IA vérifie s’il y a une victoire en appelant la fonction « victoire\_morpion » ainsi grâce au « bool victoire », la variable victoire vraie ou fausse s’il y a une victoire ou non. /Ensuite le « tab[i][j] = ‘ ‘  ; » / . Si une victoire de l'adversaire est détectée, les coordonnées x et y sont mises à jour, et la fonction « return » indique que l'IA a trouvé une opportunité de bloquer l'adversaire.

-La deuxième partie est similaire à la première mais elle permet de bloquer l’adversaire. Tout comme la première boucle les 2 boucles « for » permettent de parcourir le tableau. L’IA vérifie avec un « if » dans laquelle la condition « estLibre\_morpion » la disponibilité d’un emplacement. Ensuite l’IA avec la fonction « placer\_morpion » place la croix ou le rond.

La fonction « bool victoire = victoire\_morpion(tab, size, 5, adversaire); » l’IA vérifie si le déplacement de l’adversaire permet une victoire. /Ensuite le « tab[i][j] = ‘ ‘  ; » /. La boucle « if (victoire) { x = i; y = j; return; » permet à l’IA de voir si il y a bientôt une victoire et essaie de bloquer l’adversaire.

Si une victoire de l'adversaire est détectée, les coordonnées x et y sont mises à jour, et la fonction return qui indique que l'IA a trouvé une opportunité de bloquer l'adversaire.

Et pour finir le « do-while » permet à l’IA de choisir une position aléatoire grâce au « rand() » si il y’a aucune opportunité de blocage. Le « while » permet de boucler tant qu’une case libre soit trouvée.