**Stratégie:**

La stratégie du serpent est de d’obtenir au minimum 3 points pour ensuite se focaliser sur le bonus de multiplication qui coûte 2 points et surpuissance qui coûte 3 points afin d’obtenir le plus de point possible au départ. À partir d’un certain nombre de points sur sa tête (8) il ne va plus se focaliser sur la surpuissance afin de ne pas perdre trop de points et attaquer les autres serpents. Il faut ainsi trouver le bon moment pour que le serpent focalise les bonus et attaque. Le but de notre IA est aussi d’être constante et de toujours donner un nombre de points qui soit proche du résultat précédent. Nous avons attribué à chaque valeur de boîte (1,2 …etc) que l’on voulait prendre, une valeur. Puis une valeur à chaque distance afin de faire un ratio, puis grâce à ça, on calcule un score qui nous permet de voir si une direction est intéressante ou non. Et en enlevant en plus des points au score si un serpent plus gros que nous et dans la zone et que nous n’avons pas de surpuissance, s’il est plus petit, on priorise sa direction s’il n’a pas de protection.

**Est\_sur\_arene:**

Cette fonction a pour simple but de vérifier sur une position se trouve sur l’arène. Nous utilisons donc la fonction get\_dim() du fichier matrice.py qui permet de retourner les dimensions de la matrice mise en paramètre. Après l’avoir stocké dans une variable, on vérifie si les coordonnées données sont comprises entre 0 et les dimensions de la matrice. Cette fonction a pour complexité O(1).

**Direction\_possibles:**

Pour cette fonction, nous veillons à ce que le serpent se dirige vers les bonnes directions afin de ne pas perdre de points en fonçant sur les murs, sortant de l’arène ou allant dans une case contenant une boîte de valeur supérieure à la valeur de sa tête. Le seul cas où il peut potentiellement aller vers un mur c’est s’il a un bonus de mange-mur actif. Nous avons au préalable écrit une fonction est\_sur\_arene qui vérifie si les coordonnées données se situent dans l’arène en utilisant ses dimensions dans la fonction get\_dim. Cette fonction a pour complexité O(1).

**Get:**

Cette fonction nous est utile pour fabriquer le calque. Elle retourne si la position donnée se trouve dans l’arène la valeur de la case. Par exemple si la case contient une boîte de 1 points elle va retourner 1, une boite de 2 points 2, un bonus multiplication -2 …etc. Elle a pour complexité 0(1).

**Fabriquer\_calque:**

Cette fonction permet de fabriquer un calque de l’arène. On récupère les coordonnées du serpent et crée une variable i qui commence à 0. On affiche les murs de l’arène.Puis on parcourt l’arène par ligne et colonne et en utilisant le principe de l'inondation, on note chaque case n’étant pas un mur de la valeur i qui correspond à la distance la plus courte entre le serpent et une case. Pour faire cette fonction, nous avons utilisé celle du TP12 puis nous l’avons modifiée pour qu’elle puisse correspondre au type de la matrice que nous avons. En effet, ici les matrices sont représentées sous forme de dictionnaire et non liste. Il a fallu donc utiliser les fonctions du fichier matrice. La complexité de cette fonction va dépendre du nombre de lignes et de colonnes de l’arène car nous avons une boucle while et à l’intérieur 3 boucles for imbriqués. Mais nous sommes sur une complexité assez haute. Sa complexité est aux alentours de O(N^3)

**Objets\_voisinage:**

Cette fonction vise à retourner un dictionnaire ayant comme clé les directions (“N”, “O”, “S”, “E”) et comme valeur un tuple de 3 éléments (distance, val\_objet, prop) où distance correspond à la la distance entre le serpent et la case de l’objet, val\_objet correspond à la valeur de la boîte ou du bonus et prop correspond au propriétaire. Pour les boîtes, seules celles dont la valeur est plus petite que la valeur de la tête du serpent sont comptées. Nous utilisons aussi la compréhension pour initialiser nos dictionnaires contenant les directions possibles pour le serpent à sa position. Pour rendre le calcul plus facile concernant les directions, nous avons un dictionnaire dont les clés sont les directions et la valeur est le coefficient pour x (ligne) et y (colonne). De plus, nous devons tenir compte de la distance maximale à parcourir pour le serpent. Elle a une complexité aux alentours de O(N^3).

**Calculer\_score\_direction:**

Cette fonction calcule le score qu’aura le serpent en empruntant une certaine direction. Nous avons créé un dictionnaire val\_par\_obj qui a pour clé la valeur de l’objet (0 pour case vide …etc) et comme valeur l’importance de priorité de l’objet 5 étant la plus priorisée. Et un autre dictionnaire score\_dist qui de même a comme clé les distances en case et comme valeur, le niveau de priorité. Nous veillons à ne pas compter les boîtes ayant comme propriétaire le serpent. Le but est de retourner pour chaque direction un score qui pourra ensuite être comparé aux autres directions pour retourner celui avec le plus grand score. Ce score est représenté par un ratio de la distance par la valeur de l’objet. Elle a pour complexité O(N).

**Choix\_box:**

Cette fonction est simple mais elle est la plus importante. Elle vise à retourner le choix le plus optimal de direction pour obtenir le meilleur score en comparant les différents résultats de calculer\_score\_direction. Elle a une compléxité aux alentours de O(N^2).

Nous avons tenté de faire une fonction prioriser\_multiplication qui permettait de retourner la direction pour atteindre le bonus de multiplication le plus proche mais nous avons ensuite tout intégré dans une seule fonction calculer\_score\_direction en utilisant un système de priorité.

**Bilan personnel :**

**Joris :**

Malgré le manque d’implication d’un de mes camarades, cette SAE fut enrichissante car plus complexe que les autres faites jusque-là. De plus, créer une IA de choix était drôle et demandait de sacrément réfléchir, aussi bien de manière algorithmique que mathématique. Le plus dur a été de créer le dépôt git, car j’y est passé toute la première demi-journée pratiquement, mais ceci fut également intéressant car c’est une chose bien pratique lorsque l’on sait s’en servir.

**Erika :**

De même avis que Joris, j’ai eu l’impression de travailler seule avec lui surtout sur l’implémentation de l’IA et ses fonctions. Or de ça j’ai beaucoup aimée travailler ainsi, cela m’a permis de découvrir des choses et de chercher des solutions complexes en utilisant plusieurs méthodes et de prendre la plus optimale. La partie la plus compliquée a été de résoudre les problèmes de directions des serpents qui se prenaient les murs et donc perdaient beaucoup de points.

**Elias :**

SAE pas évidente surtout parce qu'on a pu expérimenter des choses qu’on avait presque jamais vu en TP/TD, avec une équipe de personnes qui n’étaient pas de notre groupe ce qui nous a permis de nous mettre en situation réel et coopérer afin de réaliser notre tâche.