1. Optimisation de memoire

1.a Représentation d'un Board dans un `short ( uint_least16_t )`

Un `board` est représenté par un `short` (plus précisément, un `uint_least16_t`), une manière compacte et efficace de stocker l'état d'un plateau de jeu de morpion. Chaque cellule du plateau (3x3, donc 9 cellules au total) peut avoir trois états : vide, croix (X), ou cercle (O). Ces états sont représentés en base ternaire (0, 1, 2), où chaque chiffre occupe une position dans le nombre ternaire.

- **Structure du Board**: `[Joueur à jouer][Cellule 0][Cellule 1]...[Cellule 8]`
- **Base Ternaire**: Chaque cellule et le joueur à jouer sont représentés en base ternaire.
- **Taille**: Le plateau a besoin de 10 chiffres ternaires (3^{10} = 59,049, moins que 2^{16} = 65,536, la capacité d'un `short`).
- **Fonctions Associées**: Vous avez des fonctions pour interagir avec cette structure, comme `get` , `set` , et des fonctions d'affichage.

1.b Représentation d'un Super Board dans un `{ uint64, uint64, uint8 } ( sboard_t )`

Le `sboard_t` est une structure plus complexe, utilisée pour représenter un "super plateau" de jeu. Il s'agit d'une extension du concept de morpion classique, où plusieurs petits plateaux contribuent à un plus grand plateau de jeu.
Cette structure utilise deux `uint64_t` et un `uint8_t` pour stocker l'état de l'ensemble du jeu.
Cela représente 17 octets de données, ce qui est relativement petit pour un jeu de morpion complexe. Ce qui permet un clonage de plateau de manière instantanée, et une utilisation efficace de la mémoire.

- **Structure du Super Board**:
  - **Premier et Second uint64_t**: Représentent les plateaux individuels (4 plateaux de 3x3 chacun dans le premier, et 4 plateaux de 3x3 + 1 plateau de 3x1 dans le second).
  - **uint8_t**: Stocke des métadonnées supplémentaires, comme l'état des deux dernières cellules et le plateau actuel à jouer.
- **Taille et Capacité**: Capable de stocker l'état complexe de plusieurs plateaux de jeu en morpion avec un espace mémoire relativement faible.

- **Fonctions Associées**: Comme pour le `board`, des fonctions spécifiques (`s_get`, `s_set`, etc.) sont fournies pour manipuler cette structure.

En résumé, cette approche de représentation des plateaux de jeu en morpion permet une utilisation efficace de la mémoire tout en offrant la flexibilité nécessaire pour gérer à la fois des jeux simples et complexes. Elle est également adaptée pour des algorithmes comme MinMax, utilisés pour calculer les meilleurs coups dans le jeu.

1.c. Memoisation des puissances de 3

Comme nous utilisons la base ternaire pour représenter les plateaux de jeu, nous avons besoin de calculer les puissances de 3. Pour cela, nous avons implémenté une fonction qui calcule les puissances de 3 et les stocke dans un tableau `static`. Cela permet d'éviter de recalculer les puissances de 3 à chaque fois que nous en avons besoin.

2. Implementation

2.a. tttree

Pour cet exercice, c'est assez facile d'explorer l'arbre de jeu en utilisant une approche récursive. En effet, un board de 3x3 est assez petit pour être exploré en entier, et les algorithmes de recherche comme MinMax peuvent être facilement implémentés de manière récursive.

Ensuite, nous affichons dans la console les résultats de l'exploration de l'arbre de jeu, en utilisant les methods du fichier `graph.h` et `board.h` dans le dossier `graphics` pour afficher l'arbres du jeu.

2.b. sm-refresh

Pour cet exercice, nous avons implémenté un jeu de super morpion, et un solver en utilisant l'algorithme MinMax basique. Nous avons implémenté une fonction pour afficher le jeu dans la console. Une autre pour demander le coups de l'utilisateur. Et une dernière pour calculer le coups de l'ordinateur.

Pour le solver, nous avons implémenté l'algorithme MinMax basique, avec une fonction d'évaluation qui prends en compte la victoire. Le programme est sensible aux variables d'environnement suivantes:

- `DEBUG` : Si défini, affiche les informations de debug dans la console.

- `SMPATH` : Si défini, prend un argument supplémentaire pour le chemin du fichier de sauvegarde.

2.c. sm-bot

Pour cet exercice, nous avons implémenté un solver en utilisant l'algorithme MinMax avec l'élagage Alpha-Beta. Nous sommes partie sur une approche utilisant l'algorithme de MCTS (Monte Carlo Tree Search).

Les raisons pour lesquelles nous avons choisi cette approche sont les suivantes (Nous n'avons pas eu le temps de tout implémenter):

- Possibilité de cache les résultats des simulations pour les réutiliser plus tard. (Possible car nous avons une structure qui fait 17 octets). Donc au lancement du programme, nous chargeons les caches des simulations précédentes.
- Possibilité de faire des simulations en parallèle. (À l'aide de threads) et d'optimiser le temps de calcul.

3. Conclusion

La conception et l'implémentation de ces systèmes de jeu de morpion et de super morpion démontrent une application efficace des principes de l'informatique dans la résolution de problèmes complexes avec des ressources limitées. L'optimisation de la mémoire et l'utilisation de structures de données adaptées jouent un rôle crucial dans le développement de ces jeux.

Points Clés:
Optimisation de Mémoire: L'utilisation de uint_least16_t et de structures complexes comme { uint64, uint64, uint8 } pour représenter les plateaux de jeu montre une approche ingénieuse pour minimiser l'utilisation de la mémoire tout en maintenant une représentation complète et fonctionnelle du jeu.

Efficacité Algorithmique: L'application d'algorithmes tels que MinMax et MCTS, ainsi que des techniques comme l'élagage Alpha-Beta, souligne l'importance de l'efficacité algorithmique. Ces méthodes permettent de gérer la complexité des décisions dans le jeu tout en optimisant les performances.

Extensibilité et Flexibilité: La conception permet une grande flexibilité et extensibilité. Les structures de données et les algorithmes utilisés peuvent facilement être adaptés ou étendus pour d'autres types de jeux ou d'applications nécessitant une prise de décision rapide et efficace.

Utilisation des Ressources Système: L'approche parallèle et l'utilisation de cache pour les résultats des simulations mettent en avant une utilisation judicieuse des ressources système. Cela permet une exécution plus rapide et plus efficace, en particulier dans des environnements avec des capacités de calcul limitées.

En conclusion, ce projet illustre l'importance de la conception minutieuse, de l'optimisation de la mémoire, et de l'efficacité algorithmique dans le développement de jeux informatiques. Les compétences et techniques appliquées ici peuvent servir de modèle pour d'autres projets dans des domaines similaires, soulignant l'importance de l'informatique dans la résolution de problèmes complexes