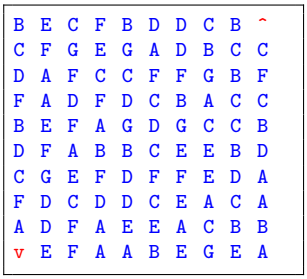
Les 7 merveilles du monde des 7 couleurs – *TP noté*

Architecture et Programmation C

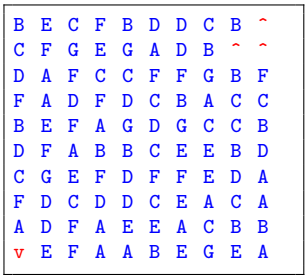
TP à rendre pour le vendredi 15 octobre 2021 avant 19h00 (CEST)
À faire en binômes

1 Règles du jeu

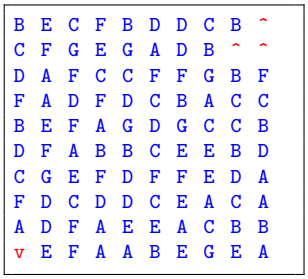
L'objectif de ce TP est d'implémenter un petit jeu de stratégie, dont voici le début d'une partie.



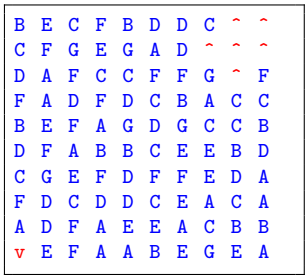
État initial



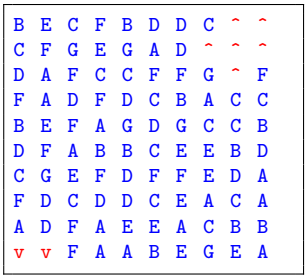
Tour 1: ^ a joué C



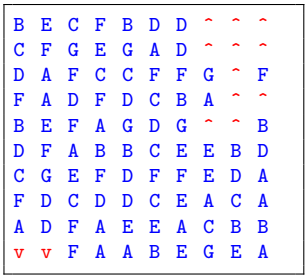
Tour 2: v a joué D



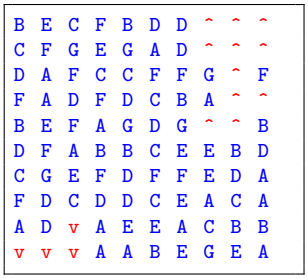
Tour 3: ^ a joué B



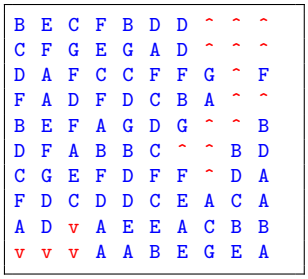
Tour 4: v a joué E



Tour 5: ^ a joué C



Tour 6: v a joué F



Tour 7: ^ a joué E

Dans ce TP, nous allons coder le merveilleux jeu des 7 couleurs. Ce jeu se passe dans un monde merveilleux qui, dans la suite, sera considéré comme étant carré avec des cases carrées. Ce monde fait très exactement 30 cases sur 30 (l'exemple ci-dessus est plus petit pour des raisons cosmétiques). Initialement, chacune des cases est aléatoirement d'une des 7 couleurs possibles (notées dans la suite A, B, C, D, E, F et G) sauf la case en haut à droite et la case en bas à gauche du monde. Ces deux cases appartiennent respectivement au joueur 1 et au joueur 2 qui vont s'affronter tout au long du jeu pour conquérir le monde. Chaque joueur a sa propre couleur: la huitième couleur (notée v) et la neuvième couleur (notée ^). Pour chaque case, les cases qui lui sont adjacentes sont (si elles existent) : la case au-dessus, la case au-dessous, la case directement à droite et la case directement à gauche.

Initialement, le joueur 1 possède donc la case en bas à gauche et le joueur 2, la case en haut à droite : ce sont leurs zones respectives. À tour de rôle, chaque joueur choisit une des 7 couleurs et sa zone s'étend à toutes les cases de cette couleur qui sont adjacentes à sa zone et récursivement jusqu'à ce qu'on ne puisse plus ajouter de cases à la zone. C'est-à-dire que tous les îlots de cases (groupes de cases adjacentes de même couleur) de la couleur choisie et qui touchent le territoire du joueur font alors partie de sa zone. Le joueur qui gagne est celui qui a la plus grande zone (en nombre de cases) à la fin de la partie.

Une démo du jeu original, datant de 1991, est disponible là : `https://youtu.be/o1Lz4EIzY9I`. Notez que dans le jeu original, la zone de chaque joueur est affichée de la couleur qu'il a choisie au tour d'avant.

Avant tout, téléchargez le code fourni : `https://mquinson.frama.io/ensr-arcsys1/TP4-7colors.tgz`

2 Organiser et tester son code

Le code fourni est déjà découpé en plusieurs fichiers, et le fichier `README` contient des instructions pour compiler le code. Le code fournit un squelette de *module* dans les fichiers `board.c` et `board.h`, que vous devrez compléter. Il est recommandé de découper son code en modules, de façon à gagner en lisibilité et à réduire la taille des différents fichiers. N'hésitez donc pas à structurer votre code en le découpant en modules, en prenant exemple sur les fichiers fournis. De plus, le module est fourni avec des exemples de tests (à la fin du fichier `board.c`). Pensez à ajouter des tests à chacun de vos modules de façon similaire, cela aide à répérer les bugs. **La qualité du découpage et des tests sera prise en compte pour l'évaluation**.

3 Voir le monde en 7 couleurs

▷ **Question 1:** Remplir chaque case du monde avec une couleur tirée aléatoirement parmi les 7 couleurs. On ne remplira pas la case en haut à droite et celle en bas à gauche.

On va maintenant écrire une fonction qui met à jour le monde après un coup d'un joueur : un joueur choisit une couleur et il faut mettre à jour sa zone dans le monde. Pour cela, on utilisera la méthode suivante : parcourir linéairement le monde jusqu'à trouver une case à mettre à jour (qui est de la couleur indiquée et dont une case adjacente est déjà dans la zone du joueur) et la mettre dans la zone. Continuer à parcourir le monde jusqu'au bout. Si j'ai modifié au moins une case en parcourant le monde, le reparcourir. S'arrêter lorsqu'on fait un parcours complet sans changement.

▷ **Question 2:** Écrire une fonction qui met à jour le monde après un coup comme expliqué ci-dessus. Comment peut-on vérifier qu'elle fait ce qu'on veut? Quel est le pire cas pour cet algorithme en nombre de parcours du monde?

▷ **Question 3:** Bonus*
Réécrire la fonction précédente de manière plus efficace (sans parcourir le monde autant de fois). Vous expliquerez votre algorithme et vous expliquerez comment vous avez testé en pratique que votre fonction renvoyait bien le résultat attendu.

4 À la conquête du monde

On va maintenant faire s'affronter deux joueurs humains dans le monde merveilleux, qui vont à tour de rôle entrer au clavier la couleur qu'ils choisissent, comme dans l'exemple illustratif de la section 1.

▷ **Question 4:** Écrire les fonctions nécessaires pour faire jouer un humain contre un humain. Quelles sont les limites de votre implémentation?

Jusqu'à maintenant, notre jeu merveilleux était sans fin. Mais, toutes les bonnes choses ont une fin.

▷ **Question 5:** Donner une condition suffisante d'arrêt de la partie (où on peut déclarer un joueur victorieux sans avoir à terminer de colorier le monde entier en 2 zones). Implémenter cette condition d'arrêt et afficher à chaque tour le taux d'occupation du monde pour chaque joueur (en pourcentage).

5 La stratégie de l'aléa

Parce qu'on n'a pas toujours une deuxième joueur sous la main, nous allons faire appel à une intelligence artificielle rudimentaire bien connue : l'aléatoire.

▷ **Question 6:** Écrire une fonction qui joue pour un joueur artificiel en choisissant à chaque tour une couleur aléatoirement parmi les 7 couleurs.

▷ **Question 7:** Écrire une fonction qui joue pour un joueur artificiel en choisissant à chaque tour une couleur aléatoirement mais parmi les couleurs qui peuvent ajouter des cases à sa zone.

6 La loi du plus fort

Nous allons essayer de faire un peu mieux que l'aléatoire en faisant un algorithme glouton.

▷ **Question 8:** Écrire un joueur artificiel qui à chaque tour choisit une couleur qui lui permet d'ajouter le maximum de cases possibles à sa zone.

Nous allons maintenant faire s'affronter dans notre monde merveilleux nos joueurs artificiels.

▷ **Question 9:** Faire s'affronter le joueur artificiel aléatoire et le joueur artificiel glouton. Comment faire pour que le combat soit équitable (ie. qu'une configuration initiale donnée n'avantage pas trop un joueur)?

Nous allons maintenant faire combattre nos deux joueurs artificiels l'un contre l'autre plusieurs fois pour voir lequel gagne le plus souvent.

▷ **Question 10:** Faire un championnat de 100 parties entre le joueur artificiel aléatoire et le joueur artificiel glouton. Quel est celui qui a le plus de victoires?

7 Les nombreuses huitièmes merveilles du monde (bonus)

Nous allons maintenant faire un joueur artificiel un peu plus sophistiqué. Nous allons essayer une stratégie qui vise à augmenter le plus possible à chaque coup le périmètre de notre zone (c'est-à-dire le nombre de cases libres qui entourent notre zone). L'idée est d'étendre le plus rapidement possible notre hégémonie sur le monde.

▷ **Question 11:** Bonus*
Implémenter le joueur artificiel hégémonique. Faire un championnat de 100 parties entre le joueur artificiel hégémoniques et le meilleur autre joueur artificiel (entre le joueur aléatoire et le joueur glouton).

On va essayer une autre stratégie : le glouton prévoyant, c'est-à-dire le glouton qui, à chaque coup, essaye toutes les combinaisons possibles de deux coups d'affilée pour voir laquelle lui ferait gagner le plus de cases. Il joue alors le premier coup de la combinaison gagnante et au prochain coup, il recalculera toutes les combinaisons possibles.

▷ **Question 12:** Bonus*
Implémenter le joueur artificiel glouton prévoyant. Quelle est la complexité de votre algorithme? Comment ferait-on si au lieu de calculer pour les deux prochains coups, on calculait pour les n prochains coups?

8 Le pire du monde merveilleux des 7 couleurs (partie bonus)

▷ **Question 13:** Bonus*
Est-ce qu'on peut construire un pire monde (à son état initial) pour un joueur artificiel donné, c'est-à-dire un monde qui rend la stratégie du joueur inefficace? Expliquer la construction (ou donner un exemple construit à la main) d'un tel monde pour chacune des stratégies (aléatoire, glouton, hégémonique, glouton prévoyant).

▷ **Question 14:** Bonus*
Implémenter un joueur artificiel hybride : il prend le meilleur de chacune des stratégies implémentées jusqu'à maintenant pour construire une stratégie qui bat les stratégies implémentées jusqu'à maintenant sur 100 parties. Expliquer l'algorithme utilisé par votre joueur artificiel.

9 Pour procrastiner (cette partie n'est pas à faire)

Voici quelques unes des nombreuses variantes qu'on aurait pu implémenter dans le monde merveilleux des 7 couleurs, mais que finalement on ne fera probablement jamais :

- faire un monde à 9 couleurs, ou plus.
- faire un monde torique (où il n'y a pas de bords).
- déclarer qu'une case normale (qui n'est pas au bord du monde) a en fait 8 voisins (les 4 cases normales et les 4 en diagonale).
- faire un monde labyrinthe (avec des murs par endroits qu'on ne peut pas traverser) et voir si notre championnat de joueurs artificiels conserve son classement.
- mettre une case en or au milieu et le premier à l'avoir dans sa zone gagne.
- tous les 10 tours, une bombe tombe au hasard sur le plateau, modifiant aléatoirement le contenu de toutes les cases dans un périmètre de 6 cases.
- faire un championnat de 100 parties du joueur humain contre chacun des joueurs artificiels.
- faire un affichage graphique plus joli. Par exemple en utilisant la bibliothèque curse, comme dans le binaire `7colors-static` inclu dans le template du projet et qui devrait être jouable sur votre machine.
- faire un monde avec des cases hexagonales.
- implémenter la stratégie optimale, et le démontrer.

Si vous avez d'autres idées d'extensions à ne surtout pas implémenter, n'hésitez pas à nous en faire part dans votre rapport (tout en restant aussi concis que possible). Nous les ajouterons à cette toNOTdo pour les années futures.

Rendu et attendus

Un rapport par binôme (ou trinôme si nombre impair) est attendu. Les rapports et le source des programmes doivent être envoyés dans une archive `tar` compressée par mail aux **deux** adresses suivantes: `remi.hutin@irisa.fr` et `martin.quinson@ens-rennes.fr`. Le rapport doit être rendu au format pdf, et l'intégralité de votre programme doit être écrit en C. Si vous avez utilisé git comme conseillé, vous pouvez donner l'adresse de ce git à la place de l'archive (en vous assurant que nous y avons accès).

Les questions marquées *Bonus** sont plus ardues ou nécessitent plus de temps, elles seront comptées en plus (i.e. points supplémentaires, troncature à 20). Elles ne bloquent pas la progression du projet. Une réponse dans le rapport expliquant la méthode envisagée sera considérée même si le code n'est pas réalisé.

Vous porterez un soin particulier à l'écriture du code. *Pensez à nettoyer pour ne pas rendre un brouillon.* Le code doit être commenté et bien écrit pour être facilement lisible. Il est rappelé que l'on écrit un programme pour que d'autres humains puissent le lire, et (accidentellement seulement) pour que les machines puissent l'exécuter. Nous compilerons votre programme avec quelques `-W???` bien choisis pour une première vérification syntaxique, mais nous ne l'exécuterons pas. Nous le lirons en revanche attentivement. Quelques conseils se trouvent sur wikipédia[1].

Votre rapport doit apporter une réponse claire et détaillée à chaque question du sujet, sans reprendre trop de code. La note finale tiendra compte à la fois de votre rapport et de votre code. Relisez-vous avant de envoyer votre travail! Votre rapport doit comporter (au moins) les parties suivantes :

Introduction : quelques mots pour présenter ce projet (ce qu'on va faire et pourquoi c'est intéressant).

Une réponse construite pour chaque question : expliquer vos observations, pourquoi ça se passe comme ça, quel est le mécanisme observé, à quoi il sert, pourquoi on se pose cette question... Soyez pédagogiques!

Synthèse : une conclusion sur vos observations, expérimentations et résultats, etc. Éventuellement, si vous en avez, des commentaires sur ce qui pourrait être amélioré pour l'année prochaine ou des idées de bonus que vous auriez voulu implémenter dans ce projet.

Bibliographie: Donnez la liste de toutes les sources (sites, livres ou individus) qui vous ont aidé, avec quelques mots de ce que vous en avez retiré. Attention, la frontière est mince entre *l'oubli* de certaines sources et le plagiat. N'oubliez rien, ne trichez pas.

Par tricher, nous entendons notamment :

- Rendre le travail de quelqu'un d'autre avec votre nom dessus ;
- Obtenir une réponse par Google™ ou autre et mettre votre nom dessus ;
- Récupérer du code et ne changer que les noms de variables et fonctions ou leur ordre avant de mettre votre nom dessus (*"moving chunks of code around is like moving food around on your plate to diguise the fact that you havn't eated all your brussel sprouts"*) ;
- Permettre à un collègue de *s'inspirer* de votre travail. Assurez vous que votre répertoire de travail n'est lisible que par vous même.

Il est plus que très probable que nous détections les tricheries s'il y en a. Chacun a son propre style de programmation, et personne ne code la même chose de la même manière. De plus, il existe des programmes très efficaces pour détecter les similarités douteuses entre copies (MOSS, `http://theory.stanford.edu/~aiken/moss/`). En cas de litige grave, seul un historique progressif de vos travaux (comme en offre git) constitue une preuve de votre innocence. *Commit soon, commit often.*

En revanche, il est conseillé de discuter du projet et d'échanger des idées avec vos collègues. Pour ne pas franchir la ligne jaune, **ne lisez jamais de code écrit par un autre groupe, et ne permettez pas aux autres groupes de lire votre code**. Vous ne pouvez rendre que du code écrit par vous-même, et vous devez détailler brievement vos sources d'inspiration sur internet dans la partie bibliographie de votre rapport. Soyez spécifique: si par exemple vous avez trouvé des réponses sur Stack Overflow, pointez les pages spécifiques utilisées.

Votre travail est à rendre pour le **vendredi 15 octobre 2021 avant 19h CEST**. **Tout retard sera sanctionné : un point en moins par heure de retard entamée**. Il n'y aura aucune extension (et vous aurez donc une vraie nuit de sommeil pour attaquer votre projet de Prog du bon pied le lendemain).

[1] Notions basiques sur la lisibilité d'un code C : `http://fr.wikibooks.org/wiki/Conseils_de_codage_en_C/Lisibilit%E9_des_sources`.