# Rapport du projet 7 couleurs

Le Dilavrec Quentin et Clément Legrand Lixon

15 octobre 2017

**Résumé**

Le jeu des 7 couleurs est un jeu vidéo de stratégie. Que nous avons réimplémenté en C. Ce rapport a pour but de répondre clairement et de façon détaillée à chacune des questions que nous avons traitées présentes dans le sujet du projet.

## 1 Introduction

Le jeu des 7 couleurs aussi appelé Filler dans sa version anglophone est un jeu video de strategie/puzzle créé par Dmitry Pashkov et pour la première fois publié par la société Gamos pour MS-DOS en 1990 ! De par son gameplay très simple , peu d'intéractions avec l'utilisateur tant du point de vue des commandes joué par le(s) utilisateur(s) que par l'affichage facilement faisable dans le shell. Le jeu est donc parfaitement adapté aux programmeurs débutants en C. Le sujet est pésenté sous forme de questions chacune portant sur une partie particulière du jeu, de l'affichage à l'interation avec l'utilisateur pour finir avec les IA.

## 2 L'affichage

Dans cette première partie, nous nous sommes intéressés à l'affichage du monde des 7 couleurs, ainsi que sa mise à jour après un coup d'un des joueurs.

### 2.1 Initialisation du plateau de jeu

*Reponse de Q1*

Pour permettre l'initialisation du plateau de jeu, il faut choisir de manière aléatoire l'une des 7 couleurs disponibles, représentées par la suite par une lettre (A, B, C, ..., G). Il suffit alors de tirer un entier aléatoire entre 1 et 7 puis de le remplacer par la lettre correspondante dans le plateu de jeu.

Pour cela nous avions dans un premier temps utilisé la fonction rand() disponible en C, qui renvoie un entier, puis on calculait le reste de sa division par 7 pour obtenir un entier entre 0 et 6.

Toutefois pour plus de lisibilié et la modularité on utilise

Listing 1 – parameters.h

```
...
#define COLORS_NUMBER 7
...
```

Listing 2 – board.h

```
...
#define RANDOM_COLOR (1 + (int)(rand() \% COLORS_NUMBER))
...
```

Puis on parcourt le tableau avec deux boucles "for" imbriquées et on attribue pour chaque cellule une valeur aléatoire avec RANDOM_COLOR

Ensuite on rempli les cases (29,0) et (0,29) pour qu'elles contiennent respectivement les symboles des joueurs 1 et 2.

A l'issue de cette initialisation, nous pouvons nous intéresser au déroulement du jeu en lui-même.

### 2.2 Mise à jour du plateau

*Reponse de Q2*

Pour pouvoir (enfin) jouer au jeu, il faut que le plateau se mette à jour automatiquement une fois que l'un des joueurs a joué une lettre. Pour ce faire on parcourt le monde linéairement jusqu'à tomber sur une lettre à la fois choisie et voisine d'un symbole joueur. Dans ce cas on remplace la lettre trouvée par un symbole joueur et on recommence le parcours du monde du début, de cette façon on est sûr de n'oublier aucune lettre lors de la mise à jour, et l'algorithme se termine bien puisqu'il n'y a qu'un nombre fini de cases dans le plateau de jeu. Pour vérifier que cette fonction affiche le résultat voulu, il suffit de la tester sur un plateau, de rentrer une lettre et de vérifier qu'il ne reste plus de lettres choisies adjacentes à un symbole joueur.

Le pire cas est obtenu lorsque le joueur se trouve dans la cellule parcouru en dernier et que le plateau ne contient qu'une seule lettre et qu'elle est choisie par un joueur. On peut illustrer ce cas avec le schéma suivant, la première image correspond à la situation initiale, chacune des images suivantes montre le tableau à la fin de la double boucle "for", ce qui correspond à l'intégration de tous les voisins adjacents qu'il est possible d'avoir.

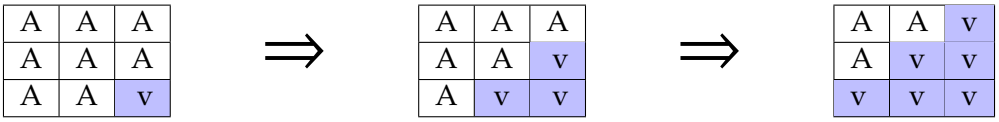


FIGURE 1 – Evolution du plateau dans le pire cas avec modify

## 3 Les interactions avec l'utilisateur

A présent nous allons développer une implémentation Humain/Humain, pour pouvoir jouer au jeu 7colors à deux.

### 3.1 De quoi jouer faire un duel entre utilisateurs

*Reponse de Q4*

Pour que deux joueurs humains puissent s'affronter dans ce jeu, il faut d'abord s'assurer que chaque joueur puisse jouer l'un après l'autre. Il suffit pour cela de choisir de manière arbitraire quel joueur commencera la partie (ici J1), puis on incrémente notre variable de un

à chaque fois qu'un joueur à joué (on repasse à J1 quand on dépasse le nombre de joueurs). Il suffit ensuite de demander au joueur une lettre à remplacer pour agrandir sa zone, puis de mettre à jour le plateau de jeu.

Toutefois il peut aussi arriver que l'un des deux joueurs tape (de manière tout à fait malencontreuse) un caractère différent de l'une des lettres attendues (c'est-à-dire une lettre majuscule (ou minuscule) entre A et G), et dans ce cas on doit renvoyer un message d'erreur adapté pour que le joueur puisse corriger.

Comme aucune condition de victoire n'a été définie, la partie entre les deux joueurs est sans fin... Il est temps d'y remédier !

### 3.2 Conditions de victoire

*Reponse de Q5*

Une première idée serait d'arrêter la partie seulement lorsque le monde est divisée en deux zones, puis de comparer le nombre de cases de chacun des joueurs. Toutefois, dès que l'un des deux joueurs possède au moins 50% du plateau, l'autre joueur ne pourra plus gagner : c'est notre condition de victoire. Ainsi la partie s'arrête dès que l'un des deux joueurs possède au moins 50% du plateau. On verifie donc à chaque tour que le pourcentage de cases possédées par chaque joueur ne dépasse pas $BOARD\_SIZE^2/PLAYERS\_NUMBER$ . Bien sûr pour ne pas faire le parcours de tout le plateau à chaque tours on stocke le nombre de cellulles possédées par chaque joueur (c'est modify qui le maintient à jour).

## 4 Les intelligences artificielles aléatoires

C'est bien de pouvoir jouer à deux, mais on n'a pas toujours d'amis sous la main, et pour pallier à ce problème nous allons nous intéresser à présent à l'implémentation de certaines Intelligences artificielles.

### 4.1 Jouer totalement aléatoirement

La première IA à programmer doit choisir aléatoirement une couleur parmi les 7 du jeu, lorsque c'est son tour. Pour ce faire il suffit de reprendre l'implémentation pour un joueur humain, sauf qu'au lieu de rentrer une couleur soi-même, elle est choisie de manière aléatoire (comme quand on rempli le tableau pour la première fois (cf Question 1)). Le plateau est ensuite mis à jour et affiché. On demande également à ce que la couleur choisie soit affichée. On peut observer, après plusieurs parties, que cette IA est particulièrement inefficace, on va donc s'intéresser à une IA aléatoire plus efficace.

### 4.2 Jouer aléatoirement plus intelligemment

*Reponse de Q7*

Cette IA, a pour objectif d'être plus efficace que l'IA précédente. On va donc s'intéresser cette à fois une IA qui choisi une couleur aléatoirment mais seulement parmi les couleurs qui peuvent agrandir sa zone.

Pour connaître les couleurs adjacentes à l'IA, on initialise un tableau de booléens B avec des "false", puis on parcourt le plateau jusqu'à tomber sur une lettre qui possède un voisin correspondant au symbole de l'IA, on transforme ensuite la lettre en un chiffre n puis le n-ième

élément de B prend la valeur "true", puis on continue le parcours du plateau. A la fin on obtient un tableau B de booléens d'indice correspondant aux couleurs, ce qui permet de connaître les couleurs adjacentes à l'IA. Il suffit ensuite de mettre les couleurs de valeur "true" dans B dans un tableau C. Ensuite on choisit un entier au hasard compris entre 0 et l'indice où on c'est arrété de mettre des couleurs dans C. Puis renvoyer la couleur lu.

Cette IA est nettement plus efficace que la précédente mais peut-on faire mieux ?

# 5 L' intelligence artificielle gloutonne

La partie précédente nous a permis de développer des intelligences artificielles basées sur l'aléatoire, mais l'aléatoire a souvent des limites. Une idée naïve pour qu'une IA gagne serait qu'à chaque tour elle essaie de récupérer le maximum de cases autour d'elle. L'objectif de cette partie est donc d'implémenter cette IA, qualifiée de gloutonne, puis de la comparer à l'IA aléatoire implémentée précédemment.

## 5.1 Maximisation du gain en cases immédiate

*Reponse de Q8*

Pour implémenter cette nouvelle IA, on construit une copie vierge du plateau de jeu (où l'on ajoute les cases du joueur courant), ainsi qu'un tableau d'entiers T, dont chaque case contiendra le nombre de cases que l'IA peut avoir en jouant la lettre associée.

On commence par parcourir le plateau de jeu, lorsqu'on tombe sur une lettre voisine d'un symbole joueur, ou voisine de la même lettre dans le plateau copie, on place cette lettre dans le plateau copie à la même place que dans le vrai plateau, on incrémente la i-ème case de T de 1 (avec i le nombre associé à la lettre trouvée) et on recommence le parcours du plateau du début tant que l'on a modifié une case durant cette boucle. A la fin, T contient le nombre de cases que l'IA pourra récupérer en jouant une lettre, il suffit alors de récupérer l'indice du maximum de T, puis l'IA jouera la couleur associée à cet indice.

Nous obtenons ainsi une IA qui cherche à maximiser son nombre de cases à chaque tour, essayons maintenant de faire jouer notre IA aléatoire améliorée et cette IA, pour savoir laquelle est la meilleure.

## 5.2 Aléatoire contre Glouton

*Reponse de Q9*

Pour ne pas désavantager une IA par rapport au l'autre, il faut qu'il y ait une répartition homogène des lettres dans le plateau (on ne doit pas avoir une lettre surreprésentée car sinon Glouton pourrait facilement être avantagée en tombant sur ce groupe de lettres, alors que pour avantager l'IA aléatoire, il faudrait qu'à chaque tour cette IA n'ait qu'un seul choix possible). Ainsi pour des plateaux totalement aléatoires de 30X30 auncune des deux IA ne peut réellement être avantagées. L'IA aléatoire améliorée ne gagne jamais contre le glouton, ce qui ne fait pas des valeurs très intéressantes à étudier.

## 5.3 Championnat

*Reponse de Q10*

En faisant en championnat de 100 parties on remarque que c'est toujours Glouton qui gagne. En effet les probabilité de victoire de l'IA aléatoire est d'environ $(1/6)^{40}$ (en effet on remarque qu'il faut environ 40 tours à deux gloutons pour finir une partie et on aura au maximum un choix entre 6 couleurs, d'où le résultat).

# 6 Conclusion

C'est un projet sympa et assez bien adapté aux débutants en C mais plus de code pour le squelette aurais peut être permis de s'intéréser plus rapidement à des points intéréssants du projet (clusterisation des elements connexes et arbre des choix possibles) et cela nous aurait aussi permis de recevoir un peu des bonnes pratiques.