INFORMATIQUE-1b

- Projet informatique -

Compte Rendu

INFO1b - Projet informatique

Lucas ROUILLÉ - Albert CROS

2024 - 2025

Question 1

create empty game state et free

```c
void groupe7_create_empty_game_state(GameState* state, int size) {
    state->map = (Color*)malloc(size * size * sizeof(Color));
    if (state->map == NULL) {
        printf("[ERROR] Allocation échouée\n");
        return;
    }
    state->size = size;
}
void groupe7_free_game(GameState* state) {
    if (state->map != NULL) {
        free(state->map);
        state->map = NULL;
    }
}
```

On utilise de la mémoire pour générer la carte et on définit une fonction free pour libérer la mémoire en fin d'exécution.

set map value

```c
void groupe7_set_map_value(GameState* state, int x, int y, Color value) {
    if (x < 0 || x >= state->size || y < 0 || y >= state->size) {
        printf("[ERROR] Coordonnée hors de la carte\n");
        return;
    }
    state->map[y * state->size + x] = value;
}
```

On initialise les coordonnées et on fixe une couleur à chaque case.

fill map

```c
void groupe7_fill_map(GameState* map) {
    if (map->map == NULL) {
        printf("[ERROR] Carte non initialisée\n");
        return;
    }
    int size = map->size;
    for (int i = 0; i < size; i++) {
        for (int k = 0; k < size; k++) {
            int valeur_aleatoire = rand() % 7;
            map->map[i * map->size + k] = (Color)valeur_aleatoire;
        }
    }
    map->map[0 * map->size + (size - 1)] = (Color)1;
```

```
    map->map[map->size * (size - 1)] = (Color)2;
}
```

On remplit la carte de valeurs aléatoires, puis on positionne les joueurs aux positions en bas à gauche et en haut à droite.

main

```
int groupe7_main(int argc, char** argv) {

    int size = atoi(argv[1]);
    if (size <= 0) {
        printf("[ERROR] Taille invalide\n");
        return 1;
    }

    srand(time(NULL));
    groupe7_create_empty_game_state(&state, size);
    groupe7_fill_map(&state);

    groupe7_set_map_value(&state, 0, size - 1, (Color)7);
    groupe7_set_map_value(&state, size - 1, 0, (Color)8);

    groupe7_print_map(&state);

    groupe7_free_game(&state);
    return 0;
}
```

On définit argv comme étant la taille de la carte on génère la carte et on mobilise la mémoire nécessaire. On affiche la carte en plaçant les joueurs 1 et 2 et on libère la mémoire.

Question 2

```

char lettres[] = {'?', '^', 'v', 'A', 'B', 'C', 'D', 'E', 'F', 'G'};

void groupe7_print_map(GameState* state) {
    if (state->map == NULL) {
        printf("[ERROR] Carte non initialisée\n");
        return;
    }
    const char* colors[] = {
        "\x1B[37m", "\x1B[37m", "\x1B[30m", "\x1B[31m",
        "\x1B[34m", "\x1B[32m", "\x1B[33m", "\x1B[35m",
        "\x1B[36m", "\x1B[37m"
    };

```

```
    printf("Carte generee (%dx%d):\n", state->size, state->size);

    for (int i = 0; i < state->size; i++) {
        for (int j = 0; j < state->size; j++) {
            Color color = groupe7_get_map_value(state, j, i);
            if (color >= 0 && color <= 9) {
                printf("%s%c\x1B[0m ", colors[color], lettres[color]);
            } else {
                printf("?");
            }
        }
        printf("\n");
    }
}
```

On parcourt ici toute la carte, on récupère la valeur et on affiche le caractère qui correspond à la valeur ainsi que la couleur associée. En cas d'erreur on affiche un '?' qui permet d'avoir tout de même un affichage pour débugger.

Question 3

Pour répondre à cette question nous avons créé une fonction auxiliaire :

```
bool groupe7_estunecasevoisine(GameState*map, int i, int k, Color car){
    int size = map->size;
    if (i==size -1 && k!= 0 && k!= size-1){
        return(map -> map[i*size+k+1]== car || map -> map[i*size+k-1]==car
            || map -> map[(i-1)*size+k]==car);
    }
    else if (i ==0){
        return(map -> map[i*size+k+1]== car || map ->map[i*size+k-1]==car||
            map ->map[(i+1)*size+k]==car);
    }
    else if (k==0){
        return(map ->map[(i+1)*size+k]== car || map ->map[(i-1)*size+k]==
            car||map ->map[(i)*size+k+1]==car);
    }
    else if (k==size -1){
        return(map ->map[(i+1)*size+k]== car || map ->map[(i-1)*size+k]==
            car||map ->map[(i)*size+k-1]==car);
    }
    else if (i==size-1 && k==0){
        return(map ->map[(i-1)*size+k]==car||map ->map[(i)*size+k+1]==car);
    }
    else if (i==size-1 && k==size-1){
        return(map ->map[(i-1)*size+k]==car||map ->map[(i)*size+k-1]==car);
    }
    else if (i==0 && k==0){
        return(map ->map[(i+1)*size+k]==car||map ->map[(i)*size+k+1]==car);
    }
```

```
    else if (i==0 && k==size -1){
        return(map ->map[(i+1)*size+k]==car||map ->map[(i)*size+k-1]==car);
    }
    else{
        return(map ->map[i*size+k+1]== car || map ->map[i*size+k-1]==car||
            map ->map[(i-1)*size+k]==car || map ->map[(i+1)*size +k]== car);
    }
}
```

Cette fonction renvoie un boléen. Elle prend en arguments deux entiers qui représentent les coordonnées d'une case de la map, la map, et une couleur car. Cette fonction va simplement regarder les voisins (haut, bas, gauche, droite) de la case mise en entrée de la fonction (de coordonnées i, k) et retourne True si elle possède un voisin de couleur car, et False sinon. Attention, il fallait faire différents cas selon l'emplacement de la case mise en entrée car par exemple si la case est sur la colonne de gauche, on ne va pas regarder son voisin de gauche pour ne pas provoquer une erreur.

```
void groupe7_miseajour_map(GameState* map,Color car, Color car2){
    int compteur = 1;
    int size = map->size;
    while (compteur!=0){
        compteur =0;
        for (int i =0; i < size ; i++){
            for (int k=0;k<size;k++){
                if(map->map[i*size +k]==car2 && groupe7_estunecasevoisine(
                    map,i,k,car)==true){
                    map->map[i*size+k]=car;
                    compteur+=1;
                }
            }
        }
    }
}
```

Comme expliqué dans le sujet, on prend trois variables en entrée, map le plateau de jeu, car la couleur du joueur, car2 la couleur que le joueur décide de jouer. On parcourt la carte de gauche à droite de haut en bas, on vient vérifier que chaque case de couleur jouée par le joueur pendant son tour est ou non une case voisine de la couleur du joueur grâce à la fonction estunecasevoisine. Si on se trouve dans cette situation pendant un parcours de la carte, on incrémente le compteur de 1, puis si à la fin du parcours de la carte, la valeur du compteur est différente de 0, on reparcours la carte pour s'assurer qu'on colorie bien les cases voisines des cases transformées. Dès que le compteur ne s'incrémente plus après un parcours de carte, le tour est fini et la carte est à jour.

Question 4

```
bool groupe7_verifie(GameState*map){
    int size = map->size;
    int compteur1=0;
    int compteur2=0;
```

```
    for (int i =0; i< size ; i++){
        for (int k=0; k<size;k++){
            if (map->map[i*size +k]==(Color)1){
                compteur1+=1;
            }
            else if (map->map[i*size +k]==(Color)2){
                compteur2+=1;
            }
        }
    }
    int moitie =size*size /2;
    if (compteur1>= moitie){
        //printf("Le joueur 1 a gagné la partie");
    }
    else if (compteur2>= moitie){
        //printf("Le joueur 2 a gagné la partie");
    }
    return(compteur1>= moitie || compteur2>= moitie);
}
```

Cette fonction permet de vérifier si un joueur a fini la partie. Pour cela, on vient créer deux compteurs, un pour chaque joueur. Puis on parcourt la map de même façon que précédemment. Dès qu'on croise une case du joueur 1, on incrémente son compteur de 1 et pareil pour le joueur 2. À la fin du parcours si un compteur est plus grand que la moitié du nombre total de cases de la carte, cela signifie que la partie est finie et que le joueur associé à ce compteur à gagné.

Question 5

Pour cette question on écris une fonction auxiliaire :

```
Color groupe7_lettre_couleur(char letter) {
    switch(letter) {
        case 'A': return (Color)3;
        case 'B': return (Color)4;
        case 'C': return (Color)5;
        case 'D': return (Color)6;
        case 'E': return (Color)7;
        case 'F': return (Color)8;
        case 'G': return (Color)9;
        case '^': return (Color)1;
        case 'v': return (Color)2;
        default: return (Color)-1;
    }
}
```

La fonction lettre_couleur prend un caractère (letter) en entrée et retourne une valeur associée à une couleur sous forme d'un type Color. Elle utilise une structure switch pour vérifier la valeur du caractère et renvoyer la couleur correspondante sous forme d'un entier. Cela permet d'associer chaque caractère à une couleur prédéfinie.

```c
int groupe7_main(int argc, char** argv) {
    srand(time(NULL));

    GameState game;
    Player player1;
    Player player2;
    char Coup1;
    char Coup2;
    int Tour = 1;

    groupe7_create_empty_game_state(&game, size);
    groupe7_fill_map(&game);
    groupe7_print_map(&game);
    groupe7_init_players(&game, &player1, &player2);

    while (groupe7_verifie(&game)==false){
        printf("Tour %d - Joueur 1 : Choisis une lettre (A, B, C, D, E, F, G):\n", Tour);
        scanf(" %c", &Coup1);
        Color couleur_choisie = groupe7_lettre_couleur(Coup1);
        groupe7_miseajour_map(&game, (Color)7, couleur_choisie);
        groupe7_print_map(&game);

        if (groupe7_verifie(&game)) break;

        printf("Tour %d - Joueur 2 : Choisi une lettre (A, B, C, D, E, F, G):\n", Tour);
        scanf(" %c", &Coup2);
        Color couleur_choisie2 = groupe7_lettre_couleur(Coup2);
        groupe7_miseajour_map(&game, (Color)8, couleur_choisie2);
        groupe7_print_map(&game);

        Tour++;
    }

    groupe7_free_game(&game);
    return 0;
}
```

Premièrement comme on utilise de l'aléatoire pour générer une carte, on utilise la commande srand(time(NULL)) pour s'assurer que le générateur de nombres aléatoires ne produira pas la même séquence de nombres à chaque exécution du programme. Ensuite, pour faire s'affronter deux joueurs humains, on vient modifier le main(). On crée une carte vide et tant que la partie n'est pas finie (condition verifie game = False) on fait jouer les deux joueurs. Pour cela on leur demande de choisir une lettre entre (A,B,C,D,E,F,G) qui représentent chacune une couleur. On enregistre cette lettre (qui est un caractère) dans la variable Coup1. Puis on transforme ce caractère en une variable Couleur_choisie cette fois-ci de type Color grâce à la fonction lettre_couleur. Ensuite, on met la map à jour et on l'affiche pour le joueur suivant à l'aide des fonctions précédentes. Si entre le tour du joueur 1 et 2 la fonction verifie passe de False à True, on sort immédiatement du while. Enfin,

on libère la carte avec free_game(&game)

Question 6

```
Color groupe7_aleatoire(){
    int valeur_aleatoire =3+ rand() % 7;
    return((Color)valeur_aleatoire);
}
```

Le joueur aléatoire est assez simple, on génère un chiffre aléatoire entre 3 et 9 inclus (ce qui correspond aux couleurs des cases disponibles sur le plateau de jeu joueurs exclus) puis on joue le coup associé à la couleur aléatoire générée. Le main() est donc légèrement modifié pour faire jouer un humain contre cette intelligence artificielle :

```
int groupe7_main(int argc, char** argv) {
    srand(time(NULL));

    GameState game;
    Player player1;
    Player player2;
    char Coup1;
    int Tour = 1;

    groupe7_create_empty_game_state(&game, size);
    groupe7_fill_map(&game);
    groupe7_print_map(&game);
    groupe7_init_players(&game, &player1, &player2);

    while (groupe7_verifie(&game)==false){
        printf("Tour %d - Joueur 1 : Choisis une lettre (A, B, C, D, E, F,
            G):\n", Tour);
        scanf(" %c", &Coup1);
        Color couleur_choisie = groupe7_lettre_couleur(Coup1);
        groupe7_miseajour_map(&game, (Color)7, couleur_choisie);
        groupe7_print_map(&game);

        if (groupe7_verifie(&game)) break;

        Color couleur_choisie2 = groupe7_aleatoire(&game, (Color)8);
        printf("Le joueur aléatoire a choisi la couleur %s\n",
            groupe7_couleurverslettre(couleur_choisie2));
        groupe7_miseajour_map(&game, (Color)8, couleur_choisie2);
        groupe7_print_map(&game);

        Tour++;
    }

    groupe7_free_game(&game);
    return 0;
```

```
}
```

Cette fois-ci, le choix de l'IA est déjà une variable de type Color donc pas besoin de faire la conversion caractère -> Color. Cependant, pour afficher le coup de l'IA, on veut faire la conversion Color-> caractère. Pour cela on utilise la fonction suivante qui prend une couleur de type Color, et retourne un pointeur vers une chaîne de caractères (const char*) qui correspond à une lettre associée à cette couleur. Cela permet de faire correctement la conversion Color -> caractère.

```
const char* groupe7_couleurverslettre(Color couleur) {
    switch (couleur) {
        case RED: return "A";
        case GREEN: return "B";
        case BLUE: return "C";
        case YELLOW: return "D";
        case MAGENTA: return "E";
        case CYAN: return "F";
        case WHITE: return "G";
        default: return "UNKNOWN_COLOR";
    }
}
```

Question 7

Pour le joueur aléatoire intelligent :

```
Color groupe7_aleatoireintelligent(GameState* map, Color joueur) {
    int size = map->size;
    bool couleurs_possibles[7] = {false, false, false, false, false, false,
        false};
    for (int i = 0; i < size; i++) {
        for (int k = 0; k < size; k++) {
            if (groupe7_estunecasevoisine(map, i, k, joueur)==true) {
                Color c = map->map[i * size + k];
                if (c >= 3 && c <= 9) {
                    couleurs_possibles[c - 3] = true;
                }
            }
        }
    }
    int compteur = 0;
    for (int i =0; i<7; i++){
        if(couleurs_possibles[i]==true){
            compteur = 1;
        }
    }
    if (compteur==0){
        int valeur_aleatoire = 3 + rand()%7;
        return (Color)valeur_aleatoire;
    }
```

```
    int valeuraleatoire = 3 +rand()%7;
    while (couleurs_possibles[valeuraleatoire-3]==false){
        valeuraleatoire = 3+ rand()%7;
    }
    return((Color)valeuraleatoire);
}
```

Cette fonction sélectionne aléatoirement une couleur parmi celles qui sont voisines des cases du joueur sur la carte. Elle parcourt d'abord toute la grille pour repérer les couleurs valides (entre 3 et 9) qui sont adjacentes à une case du joueur, puis en choisit une au hasard parmi elles. Si aucune couleur voisine n'est disponible, la fonction retourne simplement une couleur aléatoire parmi toutes les couleurs possibles.

Question 8

Pour ce joueur et les suivants, nous allons procéder de la manière suivante. On crée une ou plusieurs copies de la map, on effectue des actions sur chacune de ces copies, on compare les résultats entre ces différentes copies, puis on note le coup qui correspond au joueur avant d'effectuer ce coup sur la map initiale, puis on supprime les copies créées. Pour créer la copie d'une map, on utilise la fonction suivante :

```
GameState* groupe7_copie_map(GameState* map) {
    int size = map->size;
    GameState* copie = (GameState*)malloc(sizeof(GameState));
    copie->size = map->size;
    copie->map = (Color*)malloc(copie->size * copie->size * sizeof(Color));
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            copie->map[i * copie->size + j] = map->map[i * map->size + j];
        }
    }
    return copie;
}
```

Cette fonction alloue dynamiquement de la mémoire pour créer une nouvelle structure GameState, puis elle recopie chaque case de la carte originale vers la copie.

```
Color groupe7_glouton(GameState* map, Color joueur) {
    int liste[7]={0,0,0,0,0,0,0};
    for (int i = 3; i < 10; i++) {
        GameState* copie = groupe7_copie_map(map);
        groupe7_miseajour_map(copie, joueur,(Color)i);
        liste[i-3]=groupe7_compte_territoire(copie,joueur);
        groupe7_free_game(copie);
        free(copie);
    }
    int max = liste[0];
    int index = 3;
```

```
    for (int p=0; p<7; p++){
        if (liste[p]>max){
            max = liste[p];
            index = p+3;
        }
    }
    return (Color)index;
}
```

Pour écrire un joueur artificiel qui joue le coup qui lui fait gagner le plus de territoire à ce coup-ci, on vient créer 7 copie de la map actuelle, chaque copie va correspondre a la map au coup suivant en ayant choisi une couleur spécifique (7 couleurs différentes donc 7 copies). On vient analyser chaque copie après avoir effectué le coupe suivant en comptant le nombre de territoires dans cette copie pour le joueur en question (le glouton). Enfin, on joue la couleur qui a mené à la copie ayant le plus de territoires au coup suivant.
Pour compter les territoires, on implémente la fonction suivante qui parcourt linéairement la map et incrémente le compteur de 1 lorsqu'elle croise une case de couleur du joueur :

```
int groupe7_compte_territoire(GameState* map, Color joueur){
    int size = map->size;
    int compteur =0;
    for (int i = 0 ; i<size; i++){
        for (int k = 0; k<size ; k++){
            if (map->map[i*size +k]==joueur){
                compteur +=1;
            }
        }
    }
    return compteur;
}
```

Question 9

Pour faire s'affronter les deux IA, on modifie le main de cette façon :

```
int groupe7_main(int argc, char** argv) {
    srand(time(NULL));
    int aleatoire_victoires = 0;
    int glouton_victoires = 0;
    int size = atoi(argv[1]);
    for (int partie = 1; partie <= 500; partie++) {
        GameState game;
        Player player1;
        Player player2;
        groupe7_create_empty_game_state(&game, size);
        groupe7_fill_map(&game);
        groupe7_init_players(&game, &player1, &player2);
        int tour=0;
        while (groupe7_verifie(&game) == false) {
```

```
            tour ++;
            if (tour > 500) {
                printf("Erreur : trop de tours (%d). État actuel :\n", tour
                    );
                groupe7_print_map(&game);
                break;
            }
            Color couleur_choisie = groupe7_aleatoire();
            groupe7_miseajour_map(&game, (Color)1, couleur_choisie);
            if (groupe7_verifie(&game)) {
                aleatoire_victoires++;
                break;
            }
            Color couleur_choisie2 = groupe7_glouton(&game, (Color)2);
            groupe7_miseajour_map(&game, (Color)2, couleur_choisie2);
            if (groupe7_verifie(&game)) {
                glouton_victoires++;
                break;
            }
        }
        groupe7_free_game(&game);
    }
    printf("Résultats après 500 parties :\n");
    printf("IA Aléatoire a gagné %d parties.\n", aleatoire_victoires);
    printf("IA Glouton a gagné %d parties.\n", glouton_victoires);
    return 0;
}
```

On fait une boucle de 500 passages, dans laquelle à chaque fois on crée une nouvelle map dans laquelle s'affrontent l'IA aléatoire et l'IA glouton. Les résultats sont les suivants :

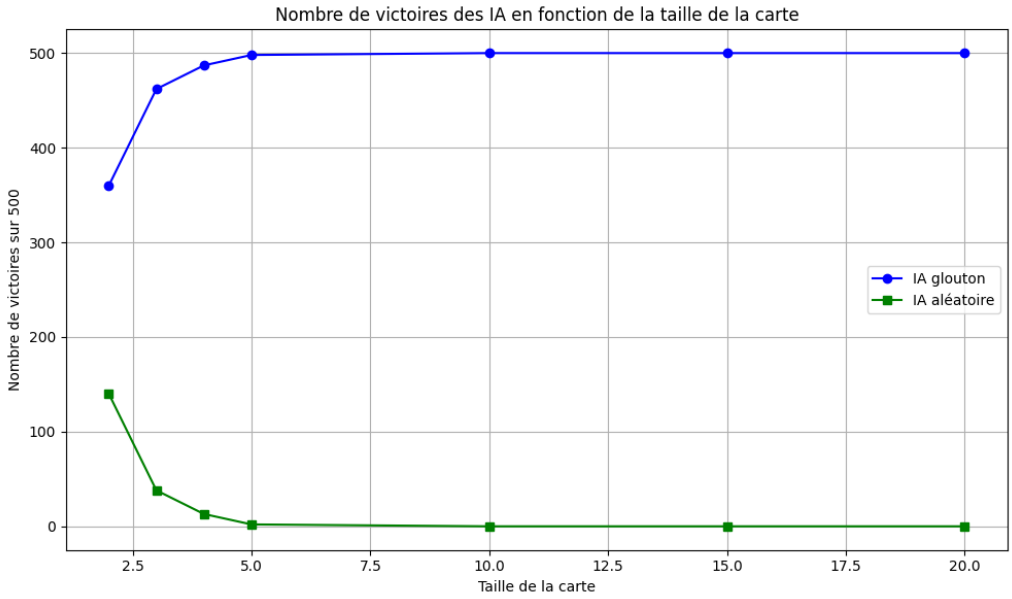

FIGURE 1 – Aléatoire vs Glouton

Question 10

Pour le joueur hégémonique, on procède de la même manière que pour le glouton c'est-à-dire qu'on crée 7 copies de la map dans lesquelles on va jouer un coup différent pour chaque copie.

```
int groupe7_compte_nombrecasesvoisines(GameState*map, Color joueur){
    int size = map-> size;
    int nb_voisins = 0;
    GameState* copie = groupe7_copie_map(map);
    for (int i=0; i<size; i++){
        for (int k = 0; k<size; k++){
            copie-> map[i*size+k]=0;
        }
    }
    for (int i=0; i<size;i++){
        for (int k=0; k<size; k++){
            if (groupe7_estunecasevoisine(map, i, k, joueur)== true ){
                if (copie->map[i*size +k]==0){
                    nb_voisins+=1;
                    copie-> map[i*size +k]=1;
                }
            }
        }
    }
    return nb_voisins;
```

```
}
```

Ensuite, on code le joueur hégémonique de la manière suivante :

```
Color groupe7_hegemonique(GameState* map, Color joueur){
    int liste[7]={0,0,0,0,0,0,0};
    for (int i = 3; i < 10; i++) {
        GameState* copie = groupe7_copie_map(map);
        groupe7_miseajour_map(copie, joueur,(Color)i);
        liste[i-3]= groupe7_compte_nombrecasesvoisines(copie, joueur);
        groupe7_free_game(copie);
        free(copie);
    }
    int max = -1;
    int index = -1;
    for (int p=0; p<7; p++){
        if (liste[p]>max){
            max = liste[p];
            index = p+3;
        }
    }
    return (Color)index;
}
```

On joue donc le coup associé à la simulation qui présente le nombre de voisins le plus grand.

Question 11

Pour le joueur mixte, on mélange les codes des deux questions précédentes. On va donc jouer comme un glouton et à chaque fois on crée une copie de la carte, sur cette copie on joue comme un glouton et si le nombre de cases du joueur n'a pas varié d'un coup à l'autre, cela signifie qu'on doit jouer de façon hégémonique.

```
Color groupe7_mixte(GameState* map, Color joueur) {
    int liste_frontieres[7] = {0, 0, 0, 0, 0, 0, 0};
    int liste_glouton[7] = {0, 0, 0, 0, 0, 0, 0};
    int max_frontieres = -1;
    int max_glouton = -1;
    int index_frontieres = -1;
    int index_glouton = -1;
    for (int i = 3; i < 10; i++) {
        GameState* copie = groupe7_copie_map(map);
        groupe7_miseajour_map(copie, joueur, (Color)i);
        liste_frontieres[i-3] = groupe7_compte_nombrecasesvoisines(copie,
            joueur);
        liste_glouton[i - 3] = groupe7_compte_territoire(copie, joueur);
        groupe7_free_game(copie);
        free(copie);
    }
    for (int p = 0; p < 7; p++) {
```

```
        if (liste_frontieres[p] > max_frontieres) {
            max_frontieres = liste_frontieres[p];
            index_frontieres = p + 3;
        }
    }
    for (int p = 0; p < 7; p++) {
        if (liste_glouton[p] > max_glouton) {
            max_glouton = liste_glouton[p];
            index_glouton = p + 3;
        }
    }
    GameState*copie2 = groupe7_copie_map(map);
    groupe7_miseajour_map(copie2, (Color)index_frontieres,joueur);
    if (groupe7_compte_territoire(copie2, joueur)==
        groupe7_compte_territoire(map, joueur)){
        return((Color)index_glouton);
    }
    else{
    return (Color)index_frontieres;
    }
}
```

Question 12

On fait s'affronter les ia sur une carte de taille 5 :

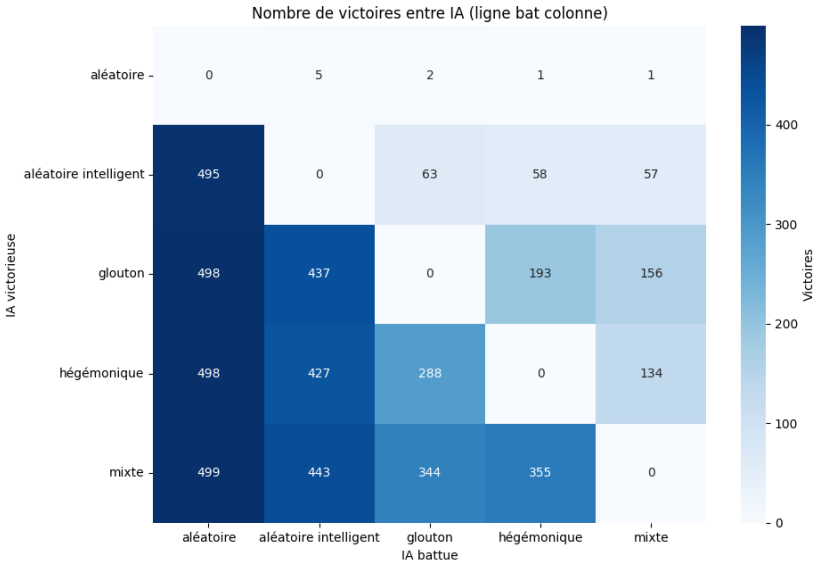

FIGURE 2 – Résultats sur une carte de taille 5

On remarque ainsi que les ia les plus performantes sont les ia mixte, glouton et hégémonique. Cependant, dans nos simulations, à chaque fois que l'on fait s'affronter nos ia, sur les 500 parties ça sera toujours la même ia qui commencera à jouer ce qui rend les duels non équitables :

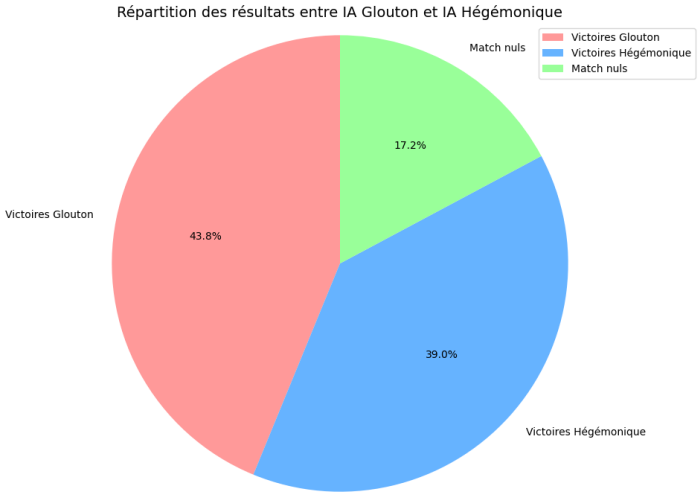

FIGURE 3 – Duels entre glouton et hégémonique sur une carte de taille 10 si glouton commence à chaque fois

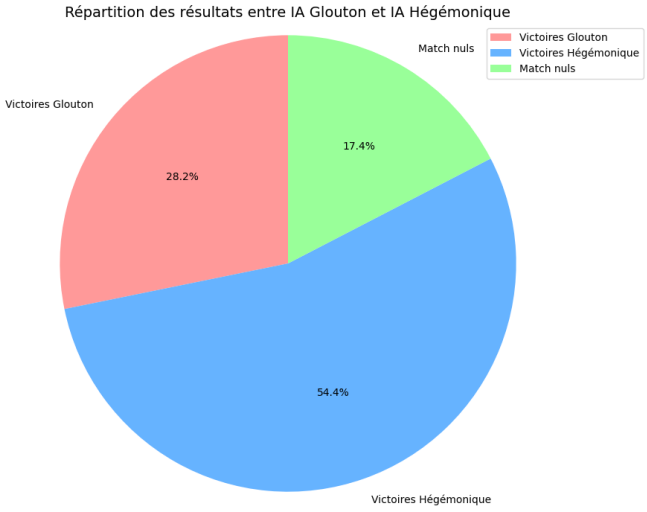

FIGURE 4 – Duels entre glouton et hégémonique sur une carte de taille 10 si hégémonique commence à chaque fois

Remarque : Sur certaines parties, l'ia hégémonique bloque la partie car elle ne peut plus

agrandir ses frontières ce qui résulte en un match nul.
Ainsi il est nécessaire d'adapter le main() afin d'alterner qui commence la partie : Si le numéro de la partie est pair l'ia 1 commence, sinon c'est l'ia 2.

```
int groupe7_main(int argc, char** argv) {
    srand(time(NULL));
    int hegemonique_victoires = 0;
    int glouton_victoires = 0;
    int size = atoi(argv[1]);
    for (int partie = 1; partie <= 500; partie++) {
        GameState game;
        Player player1;
        Player player2;
        groupe7_create_empty_game_state(&game, size);
        groupe7_fill_map(&game);
        groupe7_init_players(&game, &player1, &player2);
        int tour = 0;
        while (groupe7_verifie(&game) == false) {
            tour++;
            if (tour > 500) {
                printf("Erreur : trop de tours (%d). État actuel :\n", tour
                    );
                groupe7_print_map(&game);
                break;
            }
            if (partie % 2 == 0) {
                Color couleur_choisie = groupe7_hegemonique(&game, (Color)
                    1);
                groupe7_miseajour_map(&game, (Color)1, couleur_choisie);
                if (groupe7_verifie(&game)) {
                    hegemonique_victoires++;
                    break;
                }
                Color couleur_choisie2 = groupe7_glouton(&game, (Color)2);
                groupe7_miseajour_map(&game, (Color)2, couleur_choisie2);
                if (groupe7_verifie(&game)) {
                    glouton_victoires++;
                    break;
                }
            } else {
                Color couleur_choisie = groupe7_glouton(&game, (Color)1);
                groupe7_miseajour_map(&game, (Color)1, couleur_choisie);
                if (groupe7_verifie(&game)) {
                    glouton_victoires++;
                    break;
                }
                Color couleur_choisie2 = groupe7_hegemonique(&game, (Color)
                    2);
                groupe7_miseajour_map(&game, (Color)2, couleur_choisie2);
                if (groupe7_verifie(&game)) {
                    hegemonique_victoires++;
```

```
                    break;
                }
            }
        }
        groupe7_free_game(&game);
    }
    printf("Résultats après 500 parties :\n");
    printf("IA hégémonique a gagné %d parties.\n", hegemonique_victoires);
    printf("IA glouton a gagné %d parties.\n", glouton_victoires);
    return 0;
}
```

On obtient les résultats suivants :

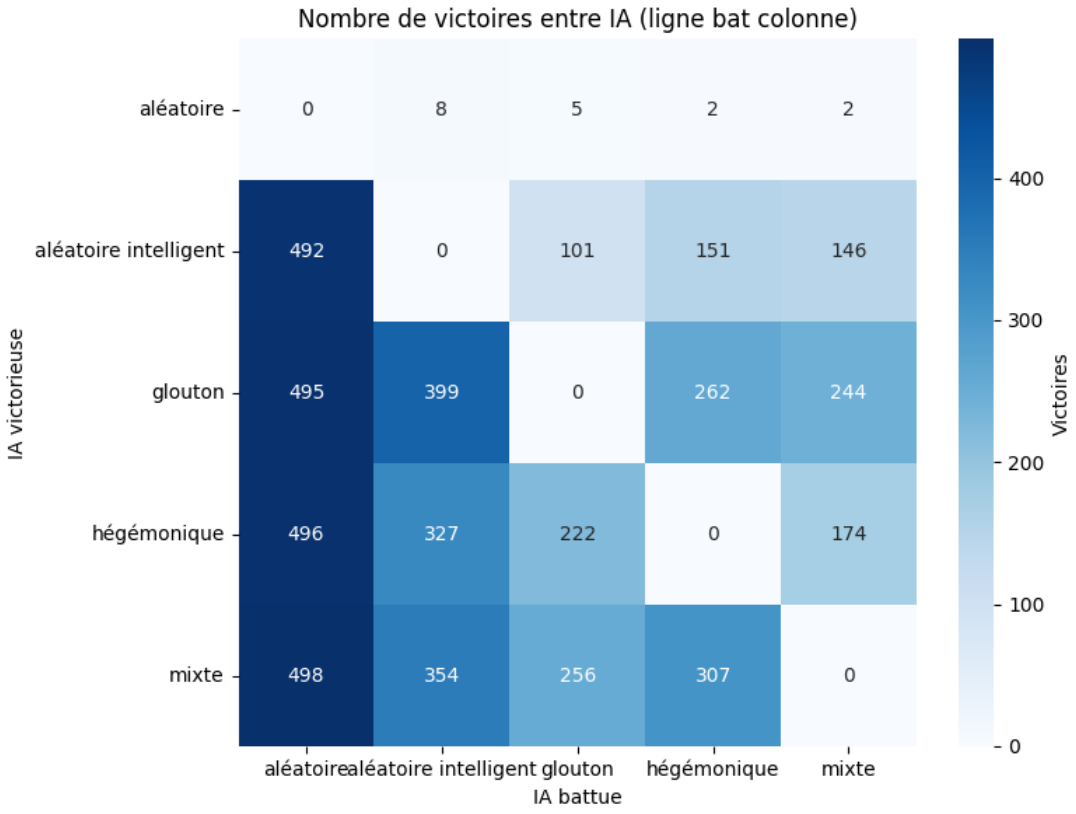

FIGURE 5 – Duels pour une carte de taille 5

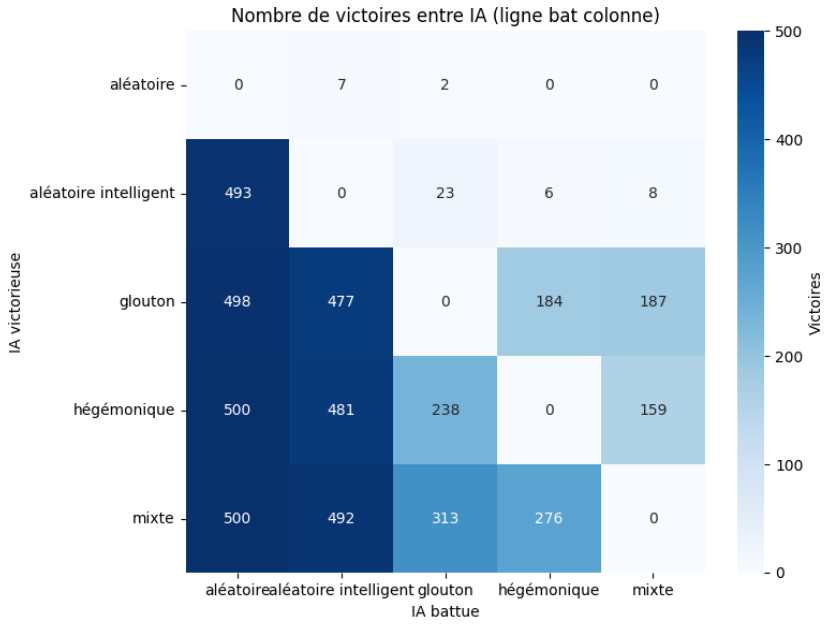

FIGURE 6 – Duels pour une carte de taille 10

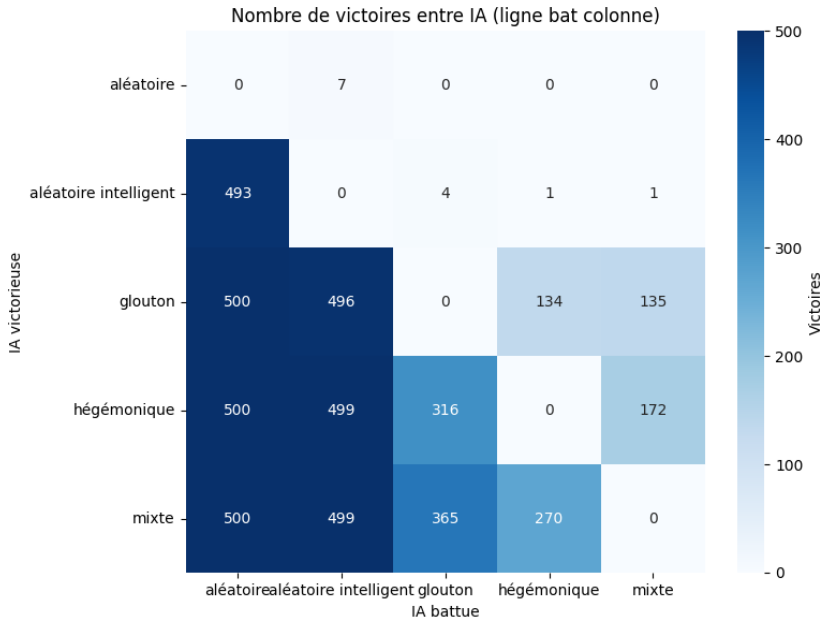

FIGURE 7 – Duels pour une carte de taille 15

Conclusion

Au terme de nos simulations, il apparaît que l'IA mixte est globalement la plus performante, quelle que soit la taille de la carte. Son approche équilibrée, combinant la maximisation du territoire et le contrôle des frontières, lui confère un avantage décisif sur les autres stratégies. Toutefois, nos résultats montrent que l'IA glouton peut surpasser l'IA hégémonique sur les petites cartes, où la prise rapide de territoire est souvent décisive. À l'inverse, sur les grandes cartes, la stratégie hégémonique devient plus efficace, car le contrôle des frontières prend une importance croissante au fil de la partie.

Amélioration

Une éventuelle perfection de l'ia mixte serait dans la façon dont nous avons codé hégémonique. Le calcul des fontières n'est pas parfait, en effet nous calculons toutes les frontières mais si des cases non converties se trouvent entourées par le joueur, ces cases ne doivent pas être considérées dans le calcul des frontières car elles ne sont pas accessibles par l'autre joueur et sont donc "safe".