#include "stdio.h"

#include "stdlib.h"

#define TAILLE 15

#define TAILLE\_MIN 6

#define REDUCTION 3

#define SIZETAIL 20

#define PLAYER 3

#define TOUR 20

int prototype\_ia (int numero\_joueur,int\*\* carte, int taille, int tour) /// IA random/controlée par utilisateur. Sert d'exemple au format des IA.

{ /// renvois un nombre entre 0 et 3 qui sert d'indication sur la direction où elle veut aller

int mvt;

printf("J %d :",numero\_joueur);

scanf("%d",&mvt);

while (mvt<0)

{

mvt++;

}

while(mvt>3)

{

mvt++;

}

///mvt=rand()%4;

return mvt;

}

/// Fonctions gestion de base des variables du jeu :

int\*\* cree\_carte (int taille1, int taille2) /// Génère une carte de format taille1 X taille2 (aucun control sur son contenu)

{

int\*\* tableau;

int i;

tableau=(int\*\*) malloc(taille1\*sizeof(int\*));

for (i=0;i<taille1;i++)

{

tableau[i]=(int\*) malloc(taille2\*sizeof(int));

}

return tableau;

}

void afficher\_carte(int\*\* carte,int taille1, int taille2) /// Affiche une carte 2D (sans couleurs) en console /// A modifier quand on aura SDL

{

int i,j;

for(i=0;i<taille1;i++)

{

for(j=0;j<taille2;j++)

{

printf(" %d",carte[i][j]);

}

printf("\n");

}

}

void initialiser\_carte(int\*\* carte,int hauteur,int largeur,int valeur) /// Remplie une carte 2D avec le nombre "valeur"

{

int i,j;

for(i=0;i<hauteur;i++)

{

for(j=0;j<largeur;j++)

{

carte[i][j]=valeur;

}

}

}

/// Fin fonctions variables du jeu ///

/// Fonctions utilitaire :

int\* position (int numero\_joueur,int\*\* carte, int taille ) /// Donne la position d'un joueur en cherchant son nombre dans la carte

{ ///(ne marche que si le joueur n'a qu'une position)

int i,j; /// renvois {-1;-1} sinon

int\* tab;

tab= (int\*)malloc(2\*sizeof(int));

tab[0]=-1;

tab[1]=-1;

for(i=0;i<taille;i++)

{

for(j=0;j<taille;j++)

{

if (carte[i][j]==numero\_joueur)

{

tab[0]=i;

tab[1]=j;

return tab;

}

}

}

return tab;

}

int max\_tab (int\* tab, int taille) /// Donne le maximum d'un tableau 1D d'entier.

{

int i;

int max=tab[0];

for (i=1;i<taille;i++)

{

if (max<tab[i])

{

max=tab[i];

}

}

return max;

}

void free\_tab\_2D (int\*\* tab,int taille) /// Sert à libérer la mémoire d'un tableau dynamique 2D.

{

int i;

for (i=0;i<taille;i++)

{

free(tab[i]);

}

free(tab);

}

int\*\* copy\_carte (int\*\* carte, int taille1, int taille2) /// Copie un tableau 2D

{

int i,j;

int\*\* copy;

copy=cree\_carte (taille1,taille2);

for (i=0;i<taille1;i++)

{

for (j=0;j<taille2;j++)

{

copy[i][j]=carte[i][j];

}

}

return copy;

}

int\* copy\_tab\_1D (int\* tab, int taille) /// Copie un tableau 1D

{

int i;

int\* copy;

copy=(int\*) malloc(taille\*sizeof(int));

for (i=0;i<taille;i++)

{

copy[i]=tab[i];

}

return copy;

}

int compt\_in\_tab (int\* tab, int taille, int a) /// Renvoie le nombre de fois qu'un nombre est présent dans un tableau.

{

int i;

int nbr=0;

for (i=0;i<taille;i++)

{

if (tab[i]==a)

{

nbr++;

}

}

return nbr;

}

/// Fin fonctions utilitaire ///

/// Fonctions gerant la descision et le déplacement des IA :

void gestion\_traine (int\*\* carte,int taille, int trainee) /// Controle la taille des trainees

{ /// Les trainees sont congru à leur joueur modulo PLAYER (remarque: le dernier joueur et ses trainees sont congru à 0)

int i,j; /// On utilisera ce modulo pour déterminer l'appartenance d'une trainee.

for(i=0;i<taille;i++)

{

for(j=0;j<taille;j++)

{

if(carte[i][j]>PLAYER\*trainee)

{

carte[i][j]=0;

}

else if(carte[i][j]>0)

{

carte[i][j]=carte[i][j]+PLAYER;

}

}

}

}

void annonce\_crashs (int\* annonce\_c, int t\_crashs, int\* statu) /// Annonce les crashs sur trainées. /// A modifier quand on aura SDL

{

int i;

for (i=0;i<t\_crashs;i++)

{

if (annonce\_c[i]==0)

{

printf("Joueur %d s'est tué sur la trainée de joueur %d\n",PLAYER,statu[0]);

}

else

{

printf("Joueur %d s'est tué sur la trainée de joueur %d\n",annonce\_c[i],statu[annonce\_c[i]]);

}

}

printf("\n");

}

void annonce\_sorties (int\* annonce\_s, int t\_sorties, int\* statu) /// Annonce les sorties de la carte. /// A modifier quand on aura SDL

{

int i;

for (i=0;i<t\_sorties;i++)

{

if (annonce\_s[i]==0)

{

printf("Joueur %d à fuis le combat\n",PLAYER);

}

else

{

printf("Joueur %d à fuis le combat\n",annonce\_s[i]);

}

}

printf("\n");

}

void saisie\_deplacement\_ia(int\*\* carte, int taille, int\* statu, int\*\* liste\_positions,int tour) /// Sauvegarde là où veulent aller les IAs. Gère les crash sur trainé et sorties de la carte.

{

int i;

int direction; /// Récupère la direction vers où se dirige l'IA.

int\*\* copy; /// Copy de la carte. Sert à éviter que les IAs ne modifie la carte pour tricher.

int\* annonce\_c; /// Sert à annocer les crashs sur trainée et les sorties de la carte.

int t\_crashs = 0;

int\* annonce\_s;

int t\_sorties = 0;

annonce\_c= (int\*) malloc(sizeof(int));

annonce\_s= (int\*) malloc(sizeof(int));

initialiser\_carte(liste\_positions,PLAYER,2,-1); /// Initialise les positions des IAs à [-1][-1] : C'est la position des morts.

for (i=0;i<PLAYER;i++)

{

if (statu[i]==0) /// Verifie si l'IA est en vie pour ne pas toucher aux morts.

{

copy=copy\_carte(carte,taille,taille); /// Copie la carte pour la donner à l'IA. Se copie pour chaque IA pour évité qu'une IA reçoive une carte rendue fausse par la précedente.

direction=prototype\_ia (i,copy,taille, tour); /// Demande la direction à l'IA.

free\_tab\_2D(copy,taille); /// Libere la mémoire de la copie.

if (i>0) /// Récupère les coordonnées actuelles de l'IA. Le if sert pour la dernière IA, stockée à l'indice 0 mais de nombre PLAYER.

{

liste\_positions[i][0]=position(i,carte,taille)[0];

liste\_positions[i][1]=position(i,carte,taille)[1];

}

else

{

liste\_positions[i][0]=position(PLAYER,carte,taille)[0];

liste\_positions[i][1]=position(PLAYER,carte,taille)[1];

}

switch (direction) /// Modifie les coordonnées de l'IA en fonction de la direction vers laquelle elle se déplace.

{

case 0:

liste\_positions[i][0]=liste\_positions[i][0]+1;

break;

case 1:

liste\_positions[i][1]=liste\_positions[i][1]+1;

break;

case 2:

liste\_positions[i][0]=liste\_positions[i][0]-1;

break;

case 3:

liste\_positions[i][1]=liste\_positions[i][1]-1;

break;

}

if (liste\_positions[i][0]<0 || liste\_positions[i][0]>=taille || liste\_positions[i][1]<0 || liste\_positions[i][1]>=taille) /// Test si l'IA sort de la carte.

{

statu[i]=-1; /// Si oui, tue l'IA et l'ajoute à la liste des annonces. statu -1 indique une mort par sortie de la carte.

liste\_positions[i][0]=-1;

liste\_positions[i][1]=-1;

t\_sorties++;

annonce\_s=(int\*)realloc(annonce\_s,t\_sorties\*sizeof(int));

annonce\_s[t\_sorties-1]=i;

}

else if (carte[liste\_positions[i][0]] [liste\_positions[i][1]]!=0) /// Test si l'IA rentre dans une trainée.

{ /// Si oui, tue l'IA et l'ajoute à la liste des annonces.

statu[i]=carte[liste\_positions[i][0]] [liste\_positions[i][1]];

while (statu[i]>PLAYER) /// Le while sert à avoir le numéro du tueur à partir de la trainée.

{

statu[i]=statu[i]-PLAYER;

}

liste\_positions[i][0]=-1;

liste\_positions[i][1]=-1;

t\_crashs++;

annonce\_c=(int\*)realloc(annonce\_c,t\_crashs\*sizeof(int));

annonce\_c[t\_crashs-1]=i;

}

}

}

annonce\_crashs(annonce\_c,t\_crashs,statu); /// Annonce des morts et libération de la mémoire des listes des morts.

annonce\_sorties(annonce\_s,t\_sorties,statu);

free(annonce\_c);

free(annonce\_s);

}

void deplacement\_ias (int\*\* liste\_positions, int\*\* carte, int taille, int\* statu) /// Modifie la carte avec la nouvelle position des joueurs. A besoin des status et position après

{ /// detction des collisions. Modifie la trainée.

int i;

for (i=1;i<PLAYER;i++)

{

if (statu[i]==0)

{

carte[liste\_positions[i][0]][liste\_positions[i][1]]=i;

}

}

if (statu[0]==0)

{

carte[liste\_positions[0][0]][liste\_positions[0][1]]=PLAYER;

}

}

/// Fin Fonctions déplacement ///

/// Fonctions gerant les collisions entre joueurs :

// PAS ENCORE TESTEES

void annonce\_collision (int\* annonce,int t\_annonce,int\* statu) /// Sert à annoncer les collision entre joueurs /// A modifier quand on aura SDL

{

int i,j;

for (i=0;i<t\_annonce;i++)

{

printf("Ces joueurs se sont rentrées dedans :\n");

for (j=1;j<PLAYER;j++)

{

if (statu[j]==annonce[i])

{

printf("Joueur %d\n",j);

}

}

if (statu[0]==annonce[i])

{

printf("Joueur %d\n",PLAYER);

}

printf("\n");

}

}

void collision\_joueurs (int\*\* liste\_positions,int\* statu) /// Tue les joueurs se rentrant dedant, annonce les collisions

{

int i; // Compteurs

int j;

int\* annonce\_crashs; /// Liste utilisée pour annoncer les crashs du tour actuelle

int t\_annonce=0; // Taille de la liste.

int groupe=max\_tab(statu,PLAYER); // Numéro du dernier groupe de collision créé. Evite de grouper deux groupe de collision différent.

if(groupe<=PLAYER) // Si max<=PLAYER, c'est qu'aucun joueur n'est entré en collision avec un autre.

{ // On donne alors la valeur PLAYER car le prochain groupe créé aura la valeur max+1 (donc PLAYER+1). Ca évite de donner les kills à un joueur.

groupe=PLAYER; // Ca évite de donner les kills à un joueur, ou d'en réssuciter un.

}

annonce\_crashs= (int\*) malloc(sizeof(int)); /// la liste n'est pas vide pour permettre l'allocation dynamique.

for(i=0;i<PLAYER;i++)

{

for(j=i+1;j<PLAYER;j++) /// Le +1 est là car sinon, le programme compare une position avec elle-même, et considère que l'IA se crash sur elle-même.

{

if(liste\_positions[i][0]==liste\_positions[j][0] && liste\_positions[i][1]==liste\_positions[j][1] && liste\_positions[i][0]!=-1) /// Le (!=-1) sert à ne pas toucher au morts.

{ /// SE SOUVENIR DE DONNER LA POSITION {-1;-1} AU MORTS.

if (statu[i]==0 && statu[j]==0) /// Si les deux joueurs sont en vie, on créer un nouveau groupe et on les y met.

{

t\_annonce++; /// On créer aussi une nouvelle annonce.

groupe++;

annonce\_crashs=(int\*)realloc(annonce\_crashs,t\_annonce\*sizeof(int));

annonce\_crashs[t\_annonce-1]=groupe;

statu[i]=groupe;

statu[j]=groupe;

}

else if (statu[i]>PLAYER && statu[j]==0) /// Si l'un fait déjà partie d'un groupe de collision, on ajoute l'autre à ce groupe

{

statu[j]=statu[i];

}

else if (statu[i]==0 && statu[j]>PLAYER)

{

statu[i]=statu[j];

}

}

}

}

annonce\_collision (annonce\_crashs,t\_annonce,statu); /// Affichage des collisions entre joueurs

free(annonce\_crashs);

}

/// Fin fonctions collisions ///

/// Fonctions reduction taille map :

void annonce\_ejection (int\* annonce\_ejec, int t\_annonce) /// Annonce les joueurs sortis du tableau lors de sa reduction /// A modifier en SDL

{

int i;

for (i=0;i<t\_annonce;i++)

{

if (annonce\_ejec[i]==0)

{

printf("Joueur %d est ejecte de la carte\n",PLAYER);

}

else

{

printf("Joueur %d est ejecte de la carte\n",annonce\_ejec[i]);

}

}

printf("\n");

}

void ejectee\_carte (int\*\* liste\_positions, int taille, int\* statu) /// Repere les joueurs ejectés de la carte et les tue. Annonce les morts.

{

int i;

int\* annonce\_ejec;

int t\_annonce=0;

annonce\_ejec=(int\*) malloc(sizeof(int));

for (i=0;i<PLAYER;i++)

{

if (statu[i]==0) /// Regarde si le joueur est vivant

{

if (liste\_positions[i][0]==0 || liste\_positions[i][0]==taille-1 || liste\_positions[i][1]==0 || liste\_positions[i][1]==taille-1) /// Regarde si il est aux limites

{

liste\_positions[i][0]==-1; /// Tue le joueur (statu -2 pour les ejections)

liste\_positions[i][1]==-1;

statu[i]=-2;

t\_annonce++;

annonce\_ejec=realloc(annonce\_ejec,t\_annonce\*sizeof(int)); /// Ajoute le joueur à l'annonce

annonce\_ejec[t\_annonce-1]=i;

}

}

}

annonce\_ejection(annonce\_ejec, t\_annonce);

free(annonce\_ejec);

}

int\*\* reduction\_carte (int\*\* carte, int taille) /// Reduit la taille de la carte.

{

int i,j;

int\*\* carte\_reduite;

carte\_reduite = cree\_carte(taille-2,taille-2);

for (i=0;i<taille-2;i++)

{

for (j=0;j<taille-2;j++)

{

carte\_reduite [i][j] = carte [i+1][j+1];

}

}

free\_tab\_2D(carte,taille);

return carte\_reduite;

}

/// Fin fonctions reduction map ///

/// Fonctions de victoire :

int\* departager\_vainqueur (int\* statu, int\* ex\_statu) /// Utilise le nombre de kills pour tenter de départager en cas d'égalité

{

int i;

int j = 0;

int compt;

int\*\* potentiel\_winner;

int\* winner;

int score;

compt=compt\_in\_tab(ex\_statu,PLAYER,0);

potentiel\_winner = cree\_carte (compt,2);

for (i=0;i<PLAYER;i++)

{

if (ex\_statu[i]==0)

{

potentiel\_winner[j][0]=i;

potentiel\_winner[j][1]=compt\_in\_tab (statu,PLAYER,i);

j++;

}

}

score=potentiel\_winner[0][1];

winner = (int\*) malloc(2\*sizeof(int));

winner[0]=1;

winner[1]=potentiel\_winner[0][0];

for (i=1;i<compt;i++)

{

if (score<potentiel\_winner[i][1])

{

winner = (int\*) realloc (winner,2\*sizeof(int));

winner[0]=1;

winner[1]=potentiel\_winner[i][0];

}

else if (score==potentiel\_winner[i][1])

{

winner[0]=winner[0]+1;

winner = (int\*) realloc (winner,(winner[0]+1)\*sizeof(int));

winner[winner[0]]=potentiel\_winner[i][0];

}

}

free\_tab\_2D(potentiel\_winner,compt);

return winner;

}

void vainqueur (int\* statu) /// Annonce le vainqueur

{

int i;

for (i=1;i<PLAYER;i++)

{

if (statu[i]==0)

{

printf("Le Joueurs %d a gagner\n",i);

}

}

if (statu[0]==0)

{

printf("Le Joueurs %d a gagner\n",PLAYER);

}

}

void egalite (int\* winner) /// Annonce les vainqueurs en cas d'égalité

{

int i;

printf("Ont finit à égalité :\n");

for (i=1;i<winner[0]+1;i++)

{

if (winner[i]==0)

{

printf("Joueur %d\n",PLAYER);

}

else

{

printf("Joueur %d\n",winner[i]);

}

}

}

int main(void)

{

int\* statu= (int\*) calloc(PLAYER,sizeof(int));

int\* ex\_statu= (int\*) calloc(PLAYER,sizeof(int));;

int\*\* carte;

int taille=TAILLE;

int trainee = SIZETAIL;

int\*\* liste\_positions;

int en\_vie;

int tour=1;

int\* winner;

carte=cree\_carte(taille,taille);

initialiser\_carte(carte,taille,taille,0);

liste\_positions=cree\_carte(PLAYER,2);

initialiser\_carte(liste\_positions,PLAYER,2,-1);

en\_vie = compt\_in\_tab (statu, PLAYER, 0);

carte[2][4]=3;

carte[3][3]=2;

carte[3][5]=1;

afficher\_carte(carte,taille,taille);

printf("Statu :\n");

afficher\_carte(&statu,1,PLAYER);

printf("positions :\n");

afficher\_carte(liste\_positions,PLAYER,2);

printf("\n");

while (tour<=TOUR && en\_vie>1)

{

printf("Tour n'%d\n",tour);

free(ex\_statu);

ex\_statu = copy\_tab\_1D (statu,PLAYER);

saisie\_deplacement\_ia (carte, taille, statu, liste\_positions,tour);

collision\_joueurs (liste\_positions, statu);

gestion\_traine (carte,taille,trainee);

deplacement\_ias (liste\_positions, carte, taille, statu);

if (tour%REDUCTION==0 && taille>TAILLE\_MIN)

{

ejectee\_carte(liste\_positions,taille,statu);

carte=reduction\_carte(carte,taille);

taille=taille-2;

}

afficher\_carte (carte,taille,taille);

printf("Statu :\n");

afficher\_carte(&statu,1,PLAYER);

printf("Ex statu :\n");

afficher\_carte(&ex\_statu,1,PLAYER);

printf("positions :\n");

afficher\_carte(liste\_positions,PLAYER,2);

tour++;

en\_vie = compt\_in\_tab (statu, PLAYER, 0);

trainee = SIZETAIL + PLAYER - en\_vie;

printf("%d joueurs restant\nTrainee de taille %d\n\n",en\_vie,trainee);

}

if (en\_vie<1)

{

winner = departager\_vainqueur(statu, ex\_statu);

egalite(winner);

printf("\nwinner :\n");

afficher\_carte(&winner,1,winner[0]+1);

}

else

{

vainqueur(statu);

}

return 0;

}