#include <iostream>

#include <random>

#include <ctime>

#include <fstream>

#include <iomanip>

#include "Game.h"

#include "stdlib.h"

#include "conio.h"

#define A\_COMPLETER void

using namespace std;

// Fonctions à écrire:

//TODO: Fonction estEgaleCase.

bool estEgaleCase(struct Case premiereCase, struct Case deuxiemeCase)

{

if (premiereCase.x==deuxiemeCase.x && premiereCase.y==deuxiemeCase.y)

{

return true;

}

else

{

return false;

}

}

//TODO: Fonction creerSerpent.

Serpent creerSerpent()

{

Case premiereCase;

premiereCase.x = 10;

premiereCase.y = 10;

Case deuxiemeCase;

deuxiemeCase.x = 10;

deuxiemeCase.y = 11;

Serpent serpent;

serpent.cases[0] = premiereCase;

serpent.cases[1]= deuxiemeCase;

serpent.longueur = 2;

serpent.direction ='d';

serpent.indiceTete=0;

return serpent;

}

void afficherSerpent(Serpent serpent)

{

cout << serpent.cases[serpent.indiceTete].x << endl;

cout << serpent.cases[serpent.indiceTete].y << endl;

cout << serpent.cases[serpent.longueur - 1].x << endl;

cout << serpent.cases[serpent.longueur - 1].y << endl;

cout << serpent.longueur << endl;

cout << serpent.direction << endl;

cout << serpent.indiceTete << endl;

char tableau[NOMBRE\_CASES][NOMBRE\_CASES];

for (int y = 0; y < NOMBRE\_CASES; y++)

{

for (int x = 0; x < NOMBRE\_CASES; x++)

{

if (y >= serpent.cases[serpent.indiceTete].y && x >= serpent.cases[serpent.indiceTete].x && y<= serpent.cases[serpent.longueur-1].y && x <= serpent.cases[serpent.longueur-1].x)

{

tableau[y][x] = '0';

cout << tableau[y][x];

}

else

{

tableau[y][x] = ' ';

cout << tableau[y][x];

}

}

cout << endl;

}

}

//TODO: Fonction calculerIndiceQueue.

Serpent calculerIndiceQueue(Serpent serpent) {

int IndiceQueue;

if (serpent.direction == 'd')

{

IndiceQueue = 0;

}

if (serpent.direction == 'a')

{

IndiceQueue = serpent.longueur - 1;

}

if (serpent.direction == 'w')

{

IndiceQueue = serpent.longueur - 1;

}

if (serpent.direction == 's')

{

IndiceQueue = 0;

}

return serpent;

}

//TODO: Fonction avancerSerpent.

Serpent avancerSerpent(Serpent serpent)

{

switch (serpent.direction)

{

case 'a':

serpent.cases[serpent.indiceTete].x--;

break;

case 'd':

serpent.cases[serpent.indiceTete].x++;

break;

case 'w':

serpent.cases[serpent.indiceTete].y++;

break;

case 's':

serpent.cases[serpent.indiceTete].y--;

break;

}

return serpent;

}

//TODO: Fonction aPerdu.

bool aPerdu(Serpent serpent)

{

Case CASE\_LIMITE;

CASE\_LIMITE.x = NOMBRE\_CASES;

CASE\_LIMITE.y = NOMBRE\_CASES;

if (serpent.cases[serpent.indiceTete].x==CASE\_LIMITE.x && serpent.cases[serpent.indiceTete].y==CASE\_LIMITE.y)

{

return true;

}

for (int i = 0; i < serpent.longueur; i++)

{

if (serpent.cases[serpent.indiceTete].x == serpent.cases[i].x && serpent.cases[serpent.indiceTete].y == serpent.cases[i].y && i != serpent.indiceTete)

{

return true;

}

else

{

return false;

}

}

}

//TODO: Fonction genererNombreAleatoire.

int genererNombreAleatoire(int a, int b) // genere des nombres entre 2 bornes

{

int nombre = (rand() % (b - a)) + a;

return nombre;

}

//TODO: Fonction genererPositionPomme.

Case genererPositionPomme(Case Position) //genere nombre entre 1 et 20

{

Position.x= (rand() % 20);

Position.y= (rand() % 20);

return Position;

}

//TODO: Fonction aMangePomme.

bool aMangePomme(Serpent serpent,Case Position)

{

if (Position.x == serpent.cases[serpent.indiceTete].x && Position.y == serpent.cases[serpent.indiceTete].y)

{

return true;

}

else

{

return false;

}

}

//TODO: Compléter l'entête de fonction.

//void allongerSerpent(Serpent serpent)

//{

// Le but est d'ajouter une nouvelle case à la queue du serpent, superposée à sa queue actuelle. Ce n'est que lors du prochain avancement qu'on verra visuellement que le serpent est plus long (i.e. que la case ne sera pas supperposée à une autre case du serpent).

//TODO: Si la tête n'est pas la première case du tableau, la queue n'est pas la dernière, et il n'est donc pas directement possible de placer une nouvelle case après la queue. Il faut:

//TODO: - insérer un espace dans le tableau à l'indice de la tête, c'est-à-dire decaler toutes les Cases dans le tableau, de la tête à la fin du tableau, vers les indices plus haut.

//TODO: - mettre à jour l'indice de la tête. La Case qui suit l'ancienne queue est maintenant libre.

//TODO: Modifier les coordonnées de la nouvelle case (nouvelle queue) pour la superposer à l'ancienne queue.

//TODO: Mettre à jour la longueur du serpent.

//}

//TODO: Fonction ecrireFichierScore.

//TODO: Fonction transcrireTableScore.

void tests()

{

//TODO: Écrire les tests.

// Test pour case identique

Case position[4];

position[0].x = 5;

position[0].y = 5;

position[1].x = 6;

position[1].y = 5;

position[2].x = 5;

position[2].y = 4;

position[3].x = 5;

position[3].y = 5;

cout << estEgaleCase(position[0], position[1])<< endl;

cout << estEgaleCase(position[0], position[2])<< endl;

cout << estEgaleCase(position[0], position[3]) << endl;

// Test pour afficher le serpent creer

Case tete;

tete.x = 10;

tete.y = 10;

Case queue;

queue.x = 10;

queue.y = 15;

Serpent serpentTest;

serpentTest.cases[serpentTest.indiceTete] =tete;

serpentTest.cases[serpentTest.longueur-1] = queue;

serpentTest.indiceTete = 0;

serpentTest.longueur = 5;

serpentTest.direction = 'a';

afficherSerpent(serpentTest);

// Test pour faire avancer le serpent

Serpent serpentM;

serpentM.cases[serpentM.indiceTete] = tete;

serpentM.cases[serpentM.longueur-1] = queue;

serpentM.indiceTete = 0;

serpentM.longueur = 5;

serpentM.direction = 'w';

avancerSerpent(serpentM);

cout << serpentM.cases[serpentM.indiceTete].x << " " << serpentM.cases[serpentM.indiceTete].y << endl;

cout << serpentM.cases[serpentM.longueur-1].x << " " << serpentM.cases[serpentM.longueur-1].y << endl;

cout << serpentM.indiceTete << endl;

cout << serpentM.longueur << endl;

cout << serpentM.direction << endl;

Serpent serpentDirectionW;

serpentDirectionW.cases[0] = tete;

serpentDirectionW.cases[4] = queue;

serpentDirectionW.indiceTete = 0;

serpentDirectionW.longueur = 5;

serpentDirectionW.direction = 'w';

Serpent serpentDirectionD;

serpentDirectionD.cases[0] = tete;

serpentDirectionD.cases[4] = queue;

serpentDirectionD.indiceTete = 4;

serpentDirectionD.longueur = 5;

serpentDirectionD.direction = 'd';

Serpent serpentDirectionS;

serpentDirectionS.cases[0] = tete;

serpentDirectionS.cases[4] = queue;

serpentDirectionS.indiceTete = 4;

serpentDirectionS.longueur = 5;

serpentDirectionS.direction = 's';

Serpent serpentDirectionA;

serpentDirectionA.cases[0] = tete;

serpentDirectionA.cases[4] = queue;

serpentDirectionA.indiceTete = 0;

serpentDirectionA.longueur = 5;

serpentDirectionA.direction = 'a';

}

// Ne pas modifier le main sauf pour commenter et decommenter certaines parties pour faire vos tests.

int main()

{

srand(unsigned(time(nullptr)));

tests();

#if false

const string NOM\_FICHIER\_SCORES = "historique.txt";

Game jeu;

Serpent serpent = creerSerpent();

definirCasesSerpent(jeu, serpent);

Case pomme = { 5, 5 };

definirCasePomme(jeu, pomme);

double tempsEntreAvancementsSerpent = 0.2;

const double accelerationAChaquePoint = 0.98; // Facteur appliqué sur le temps entre avancements, pour accélérer le jeu à chaque pomme mangée.

int score = 0;

Clock horloge;

// Boucle du jeu.

while (fenetreEstOuverte(jeu) && !aPerdu(serpent)) {

verifierEvenements(jeu);

changerDirection(serpent);

if (tempsEcouleEnSecondes(horloge) > tempsEntreAvancementsSerpent) {

avancerSerpent(serpent);

if (aMangePomme(serpent, pomme)) {

allongerSerpent(serpent);

genererPositionPomme(pomme);

tempsEntreAvancementsSerpent \*= accelerationAChaquePoint;

score++;

}

afficher(jeu);

remettreAZero(horloge);

}

}

cout << "Partie terminee! Vous avez obtenu " << score << " points." << endl;

ecrireFichierScore(NOM\_FICHIER\_SCORES, score);

int tableScore[N\_MAXIMUM\_SCORES];

unsigned nScores = transcrireTableScore(NOM\_FICHIER\_SCORES, tableScore);

int meilleurScore = 0;

for (unsigned i = 0; i < nScores; i++)

if (tableScore[i] > meilleurScore)

meilleurScore = tableScore[i];

cout << "Le meilleur score est " << meilleurScore << " points." << endl;

if (score == meilleurScore)

cout << "Bravo!" << endl;

#endif

}