```
/*
-----------------------------------------------------------------------------
Nom du fichier      : main.cpp
Auteur(s)           : Baume Oscar & Centeno Cédric
Date creation       : 14.01.2022
Description         : Programme qui simule une bataille entre un nombre choisi par
                      l'utilisateur de robot, dans un plateau au dimension choisi
                      par l'utilisateur.
Remarque(s)         : Il faut un minimum de 2 robots pour avoir une bataille.
Modification:       ---
                    Date   :
                    Auteur :
                    Raison :
Compilateur         : Mingw-w64 g++ 11.2.0
-----------------------------------------------------------------------------
*/
#include <iostream>
#include <limits>
#include "Plateau.h"
#include "saisie.h"

using namespace std;

int main() {
   const string MSG_BIENVENUE = "Bonjour, ce programme va simuler une \'bataille\'"
                                " entre un nombre choisi de \nrobot dans un "
                                "plateau d'une dimension de votre choix. Les robots se "
                                "deplace \naleatoirement sur le plateau, lorsqu\'un "
                                "robot se deplace sur une case deja occupe\nle "
                                "robot occupant la case est tue.",
                MSG_FIN       = "Appuyez sur ENTER pour mettre fin au programme.",
                MSG_ERREUR    = "Veuillez entrer une valeur correct";

   const int    MIN_DIMENSION = 10,  MAX_DIMENSION = 1000,
                MIN_ROBOT     = 1,   MAX_ROBOT     = 10;

   cout << MSG_BIENVENUE << endl << endl;
   // création du tableau comprenant la saisie contrôlée des valeurs
   Plateau plateau(saisie_entre(MIN_DIMENSION, MAX_DIMENSION,
                                "Largeur",
                                MSG_ERREUR),
                   saisie_entre(MIN_DIMENSION,MAX_DIMENSION,
                                "Hauteur",
                                MSG_ERREUR),
                   saisie_entre(MIN_ROBOT,MAX_ROBOT,
                                "Nombre objet",
                                MSG_ERREUR));
   // affichage du plateau avant que le jeu commence
   cout << plateau;
   // lance la bataille
   plateau.jouer();
   // affiche le message de fin de jeu
   cout << MSG_FIN << endl;
   cin.ignore(numeric_limits<streamsize>::max(), '\n');
}

/*
-----------------------------------------------------------------------------
Nom du fichier      : preferences.h
Auteur(s)           : Baume Oscar & Centeno Cédric
Date creation       : 14.01.2022
Description         : En-tête mettant à disposition des allias permettant de
                      modifier les types de données à un seul endroit
Remarque(s)         :
Modification:       ---
                    Date   :
                    Auteur :
                    Raison :
Compilateur         : Mingw-w64 g++ 11.2.0
-----------------------------------------------------------------------------
*/
```

```
#ifndef CPP_PREFERENCES_H
#define CPP_PREFERENCES_H

#include <utility>

using Data = unsigned;
using Coord = std::pair<Data,Data>;

#endif //CPP_PREFERENCES_H

/*
--------------------------------------------------------------------------------
Nom du fichier      : Robot.h
Auteur(s)           : Baume Oscar & Centeno Cédric
Date creation       : 14.01.2022
Description         : En-tête de la class Robot.
                      Définition de la class et de ces composants
Remarque(s)         :
Modification:       ---
                    Date   :
                    Auteur :
                    Raison :
Compilateur         : Mingw-w64 g++ 11.2.0
--------------------------------------------------------------------------------
*/

#ifndef CPP_ROBOT_H
#define CPP_ROBOT_H

#include <utility>
#include "Plateau.h"
#include "preferences.h"

// directions utilisées pour déplacer un robot
enum class Direction {HAUT,BAS,GAUCHE,DROITE};

class Robot {
   /**
    * Nom                : operator<<
    * Définition         : Opérateur de flux pour un objet de la class Robot
    * Remarques          :
    * @param os          : Flux
    * @param robot       : Objet de la class Robot à afficher
    * @return            : On retourne le flux os
    */
   friend std::ostream& operator <<(std::ostream& os, const Robot& robot);
public:
   /**
    * Nom                : Robot
    * Définition         : Constructeur vide d'objet de la class Robot
    * Remarques          :
    */
   Robot();

   /**
    * Nom                : Robot
    * Définition         : Constructeur d'objet par copie de la class Robot
    * Remarques          :
    * @param robot       : Robot à copier
    */
   Robot(const Robot& robot);

   /**
    * Nom                : Robot
    * Définition         : Constructeur d'objet de la class Robot
    * Remarques          :
    * @param coord       : Coordonnées du Robot sous forme de pair
    */
   Robot(Coord coord);

   /**
    * Nom                : ~Robot
```

```
    * Définition         : Destructeur d'objet de la class Robot
    * Remarques          :
    */
   ~Robot();

   /**
    * Nom                : operator<
    * Définition         : Opérateur de comparaison "plus petit que" de la class
    *                      Robot
    * Remarques          :
    * @param robot       : Objet robot que l'on souhaite comparer
    * @return            : Si le robot courant est plus petit que le robot en
    *                      paramétre
    */
   bool operator<(const Robot& robot) const;

   /**
    * Nom                : operator==
    * Définition         : Opérateur de comparaison "est égal" de la class Robot
    * Remarques          :
    * @param robot       : Robot que l'on souhaite comparer
    * @return            : Si le robot courant est égal au robot en paramétre
    */
   bool operator==(const Robot& robot) const;

   /**
    * Nom                : operator=
    * Définition         : Opérateur d'affectation de la class Robot
    * Remarques          :
    * @param robot       : Robot que l'on souhaite comparer
    * @return            : Le robot courant modifié
    */
   Robot& operator=(const Robot& robot);

   /**
    * Nom                : get_num()
    * Définition         : Fonction publique permettant d'obtenir la valeur de la
    *                      donnée NO d'un Robot
    * Remarques          : Utilisé pour l'opérateur de flux de Plateau
    */
   Data get_num() const;

   /**
    * Nom                : get_coord()
    * Définition         : Fonction publique permettant d'obtenir la valeur de la
    *                      donnée coord d'un Robot
    * Remarques          :
    */
   Coord get_coord() const;

   /**
    * Nom                : deplacer
    * Définition         : Fonction qui déplace dans une des 4 directions de la
    *                      class Direction
    * Remarques          :
    * @param direction   : Dans quelle direction le robot se déplace
    */
   void deplacer(Direction direction);

   /**
    * Nom                : deplacer
    * Définition         : Fonction qui change les coordonnées du robot courant
    *                      par la pair entree en paramétre
    * Remarques          :
    * @param coord       : Coordonnées que l'on attribue a notre robot courant
    */
   void deplacer(Coord coord);
private:
   // Numéro unique de l'objet Robot
   const  Data NO;
   // Numéro de série du robot suivant
   static Data suivant;
```

```
   // Coordonnées du Robot
   Coord coord;
};

#endif //CPP_ROBOT_H

/*
--------------------------------------------------------------------------------
Nom du fichier      : Robot.cpp
Auteur(s)           : Baume Oscar & Centeno Cédric
Date creation       : 14.01.2022
Description         : Définition des fonction de la classe Robot
Remarque(s)         :
Modification:       ---
                    Date   :
                    Auteur :
                    Raison :
Compilateur         : Mingw-w64 g++ 11.2.0
--------------------------------------------------------------------------------
*/

#include <iostream>
#include "Robot.h"
#include "Aleatoire.h"

Data Robot::suivant = 0;

using namespace std;

std::ostream &operator<<(std::ostream &os, const Robot& robot) {
   return os << robot.coord.first << ", " << robot.coord.second;
}

Robot::Robot(): NO(suivant) {
   // on décide qu'un robot a par défaut les coordonnées 0 0
   coord.first = coord.second = 0;
   ++suivant;
}

Robot::Robot(const Robot& robot): NO(robot.NO) {
   // suivant n'est pas incrementé car ce constructeur est utilisé pour décaler
   // des robots dans un vecteur
   coord = robot.coord;
}


Robot::Robot(Coord coord): NO(suivant) {
   this->coord = coord;
   ++suivant;
}

Robot::~Robot() {
}

bool Robot::operator<(const Robot& robot) const{
   // comparer la première coordonnée des 2 robots
   if(coord.first < robot.coord.first) {
      return true;
   }
   else if(robot.coord.first < coord.first) {
      return false;
   }
   // si elles sont égales, comparer la deuxieme coordonnée
   else if(coord.first == robot.coord.first) {
      if (coord.second < robot.coord.second) {
         return true;
      } else {
         return false;
      }
   }
   // ce cas ne devrait jamais se produire, mais ajouté par securité
   return false;
```

```
}

bool Robot::operator==(const Robot& robot) const{
   // vérifie si deux robots différents ont les mêmes coordonnées
   return (coord == robot.coord and NO != robot.NO);
}

Robot &Robot::operator=(const Robot& robot) {
   // cast en reference car NO est une constante
   (Data&) this->NO = robot.NO;
   this->coord = robot.coord;
   return *this;
}

void Robot::deplacer(Direction direction) {
   switch (direction) {
      case Direction::HAUT   :
         coord.first -= 1;
         break;
      case Direction::BAS    :
         coord.first += 1;
         break;
      case Direction::GAUCHE :
         coord.second  -= 1;
         break;
      case Direction::DROITE :
         coord.second  += 1;
         break;
   }
}

void Robot::deplacer(Coord coord) {
   this->coord = coord;
}

Data Robot::get_num() const {
   return NO;
}

Coord Robot::get_coord() const{
   return coord;
}

/*
-----------------------------------------------------------------------------
Nom du fichier      : Plateau.h
Auteur(s)           : Baume Oscar & Centeno Cédric
Date creation       : 14.01.2022
Description         : En-tête de la class Plateau.
                      Définition de la class et de ces composants.
Remarque(s)         : On défini les class Robot et Direction vide car on en a besoin
                      dans la class pour pouvoir définir un std::vector<Robot> et
                      la fonction peut_se_deplacer().
Modification:       ---
                    Date   :
                    Auteur :
                    Raison :
Compilateur         : Mingw-w64 g++ 11.2.0
-----------------------------------------------------------------------------
*/

#ifndef CPP_PLATEAU_H
#define CPP_PLATEAU_H

#include <vector>
#include <ostream>
#include "Robot.h"
#include "preferences.h"

// pré instanciation des classes de Robot.h
class Robot;
enum class Direction;
```

```
// Status que peut avoir un plateau
enum class Status{EN_COURS, FINI};

class Plateau {
   /**
    * Nom            : operator<<
    * Définition     : Opérateur de flux pour un objet de la class Plateau
    * Remarques      :
    * @param os      : Flux
    * @param plateau : Objet de la class plateau à afficher
    * @return        : On retourne le flux os
    */
   friend std::ostream& operator<<(std::ostream& os, const Plateau& plateau);
public:
   /**
    * Nom                : Plateau
    * Définition         : Constructeur d'objet de la class Plateau
    * Remarques          : Constructeur vide n'existe pas
    * @param largeur      : Largeur du plateau
    * @param hauteur      : Hauteur du plateau
    * @param nombre_robot : Nombre de robot qui seront sur le plateau
    */
   Plateau(Data largeur, Data hauteur, Data nombre_robot);

   /**
    * Nom        : ~Plateau
    * Définition : Destructeur d'objet de la class Plateau
    * Remarques  :
    */
   ~Plateau();

   /**
    * Nom        : jouer
    * Définition : fonction qui gère la partie
    * Remarques  :
    */
   void jouer();
private:
   /**
    * Nom        : effectuer_tour
    * Définition : Fonction qui effectue un "tour" de jeu.
    *              On y déplace les robots, vérifie si il y a 2 robots à la même
    *              position -> si c'est le cas "tue" le robot qui s'est fait
    *              marcher dessus.
    * Remarques  : Fonction non const car on modifie le vecteur robots
    */
   void effectuer_tour();

   /**
    * Nom        : est_fini
    * Définition : Fonction qui retourne si la "partie" est fini
    * Remarques  :
    * @return    : Si le status du plateau est Status::FINI
    */
   bool est_fini() const;

   /**
    * Nom          : trouver_robot
    * Définition   : Fonction qui retourne les robots se trouvant sur une ligne du
    *                plateau.
    * Remarques    :
    * @param ligne : Numéro de la ligne ou il faut chercher les robots
    * @return      : Vecteur des robots de la ligne "ligne"
    */
   std::vector<Robot> trouver_robot(Data ligne) const;

   /**
    * Nom               : peut_se_deplacer
    * Définition        : fonction qui retourne si il est possible de se déplacer depuis
    *                     une coordonées dans une direction
    * Remarques         :
```

```
    * @param c          : la coordonnée qui est testée
    * @param direction  : direction dans laquel on veut savoir si il est possible
    *                     de se déplacer.
    * @return           : si le déplacement est possible
    */
   bool peut_se_deplacer(const Coord& c, Direction direction) const;

   // Vecteur des robots du plateau
   std::vector<Robot> robots;
   // Status du plateau
   Status status;
   // Dimension du plateau
    const Data  max_hauteur,
                max_largeur,
                min_hauteur,
                min_largeur;
};

#endif //CPP_PLATEAU_H

/*
--------------------------------------------------------------------------------
Nom du fichier      : Plateau.cpp
Auteur(s)           : Baume Oscar & Centeno Cédric
Date creation       : 14.01.2022
Description         : Définition des fonction de la classe Plateau
Remarque(s)         :
Modification:       ---
                    Date   :
                    Auteur :
                    Raison :
Compilateur         : Mingw-w64 g++ 11.2.0
--------------------------------------------------------------------------------
*/
#include <ostream>
#include <algorithm>
#include <iostream>
#include <iomanip>
#include <string>
#include <chrono>
#include <thread>
#include "Plateau.h"
#include "Aleatoire.h"

using namespace std;

const chrono::duration SLEEPING_TIME = 200ms;

Plateau::Plateau(Data largeur, Data hauteur, Data nombre_robot)
   : max_hauteur(hauteur), max_largeur(largeur),
     min_hauteur(0), min_largeur(0){
   status = Status::EN_COURS;
   robots.resize(nombre_robot);
   vector<Coord> temp;
   // création des coordonnées unique pour les robots
   for(Data i = 0; i < nombre_robot; ++i){
      Coord c;
      // on tourne
      do
      {
         c = {
            aleatoire((int)min_hauteur,(int)max_hauteur-1),
            aleatoire((int)min_largeur,(int)max_largeur-1)
         };
      }
         // tant que la coordonnée c se trouve déjà dans le vecteur temp
      while(find(temp.begin(),temp.end(),c) != temp.end());
      // ajouter la coordonnée c dans le vecteur temp
      temp.push_back(c);
   }

   auto i = robots.begin();
```

```
   // pour chacune des coordonnées unique
   for(Coord& c : temp){
      // on déplace le robot se trouvant à l'itérateur i
      i->deplacer(c);
      ++i;
   }
}

Plateau::~Plateau() {
   // supprimer les robots du vecteur
   robots.clear();
}

void Plateau::jouer() {
   // fait
   do{
      this_thread::sleep_for(SLEEPING_TIME);
      // raffraichi le terminale de la console
      system("cls");
      this->effectuer_tour();
      cout << *this;
   }
   // tant que la partie est en cours
   while(!this->est_fini());
}

bool Plateau::est_fini() const {
   return status == Status::FINI;
}


void Plateau::effectuer_tour() {
   // si le status est en cours et qu'il reste + 1 robot sur le plateau
   if (robots.size() != 1 and status == Status::EN_COURS) {
      // pour chaque robots du plateau
      for (auto y = robots.begin(); y < robots.end(); ++y){
         // on le délpace
         Direction direction = Direction(aleatoire((int)Direction::HAUT,
                                                   (int)Direction::DROITE));
         if(peut_se_deplacer(y->get_coord(), direction))
            y->deplacer(direction);
         for(auto i = robots.begin(); i < robots.end(); ++i) {
            // si ce sont 2 robots différent et qu'ils ont les mêmes coord
            if (y->get_num() != i->get_num()
                and y->get_coord() == i->get_coord()){
               // on supprime le robot qui se fait marcher dessus
               robots.erase(i);
            }
         }
      }
      // Si il ne reste plus qu'un robot
      if(robots.size() == 1)
         status = Status::FINI;
   } else
      status = Status::FINI;
}

vector<Robot> Plateau::trouver_robot(Data ligne) const {
   vector<Robot> output;
   // pour tour les robots du plateau
   for(const Robot& r : robots){
      // qui sont à la ligne cherchée
      if(r.get_coord().first == ligne){
         // on les ajoutes à ce qu'on va retourner
         output.push_back(r);
      }
   }
   return output;
}

ostream& operator <<(ostream &os, const Plateau& plateau) {
   // string remplis d'espace pour la largeur du plateau
```

```
   const string   ESPACE          = string(plateau.max_largeur, ' '),
      BORDURE_PLATEAU = string(plateau.max_largeur + 2, '-');
   const unsigned ZERO_ASCII      = 48;

   os << BORDURE_PLATEAU << endl;
   vector<Robot> ligne;
   for (Data i = 0; i < plateau.max_hauteur; ++i) {
      // on cherche les robots de la ligne i
      ligne = plateau.trouver_robot(i);
      // affiche la borne de gauche
      os << '|';
      // si la ligne est vide
      if (ligne.empty()) {
         // on affiche une ligne vide
         os << ESPACE;
      }
         // sinon
      else {
         // on copie une ligne vide
         string output = ESPACE;
         // et pour chaque robot de la ligne
         for (const Robot& r: ligne) {
            // on place son numéro à la coordonnées Y du robot
            output.at(r.get_coord().second) = char(r.get_num() + ZERO_ASCII);
         }
         os << output ;
      }
      // affiche la borne de droite
      os << '|' << endl;
   }
   os << BORDURE_PLATEAU << endl << endl;

   if(plateau.status == Status::FINI){
      cout << "Notre vainqueur est le robot #" << plateau.robots.at(0).get_num() << endl;
   }

   return os;
}

bool Plateau::peut_se_deplacer(const Coord& coord, Direction direction) const{
   return
      (direction == Direction::HAUT    and coord.first   != min_hauteur)    or
      (direction == Direction::BAS     and coord.first   != max_hauteur -1) or
      (direction == Direction::GAUCHE  and coord.second  != min_largeur)    or
      (direction == Direction::DROITE  and coord.second  != max_largeur -1);

}

/*
-------------------------------------------------------------------------------
Nom du fichier : saisie.h
Auteur(s)      : Baume Oscar & Dorian Gillioz
Date creation  : 10.11.2021
Description    : En-tête de la librairie pour les saisies utilisateurs du programme
Remarque(s)    : ---
Modification   :
                 Date        : 15.01.2022
                 Auteur      : Oscar Baume
                 Description : Utilisation de preferences.h pour Data à la place
                               de int.
Compilateur    : Mingw-w64 g++ 11.2.0
-------------------------------------------------------------------------------
*/

#ifndef CPP_SAISIE_H
#define CPP_SAISIE_H

#include <cstdlib>
#include <string>
#include "preferences.h"

// Nom           : saisie_entre
```

```
// Description   : Permet de saisir entre deux valeurs passées en paramètres
// Remarque(s)   : Les saisies sont vérifiées et affiche un message en cas d'erreur
// Param min     : La valeur la plus petite de l'intervalle
// Param max     : La valeur la plus grande de l'intervalle
// Param message : La valeur la plus grande de l'intervalle
// Retour        : la saisie de l'utilisateur
// Exception     : n/a
Data saisie_entre(const Data& min, const Data& max, const std::string& message,const
std::string& msg_erreur);

// Nom               : recommencer
// Description       : Permet de donner la possibilité de recommencer le jeu en fonction de la réponse
donnée
// Param option_vrai : Le caractère à choisir pour redémarrer le jeu
// Param option_faux : Le caractère à choisir pour terminer le jeu
// Retour            : Si le jeu doit redémmarer ou non
// Exception         : n/a
bool recommencer(const std::string& message, const char& option_vrai,
                 const char& option_faux);

#endif //CPP_SAISIE_H

/*
-------------------------------------------------------------------------------
Nom du fichier : saisie.cpp
Auteur(s)      : Oscar Baume & Dorian Gillioz
Date creation  : 10.11.2021
Description    : Définition des fonctions de la librairie saisie
Remarque(s)    : ---
Modification   : ---
Compilateur    : Mingw-w64 g++ 11.2.0
-------------------------------------------------------------------------------
*/

#include <iostream>
#include <string>
#include <limits>
#include "saisie.h"

using namespace std;

Data saisie_entre(const Data& min, const Data& max, const string& message,const
string& msg_erreur){

   Data  saisie;
   bool erreur;

   do {
      cout << message << " [" << min << " - " << max << "] : ";
      cin >> saisie;
      erreur = cin.fail() || saisie < min || saisie > max;

      if (erreur){
         cout << msg_erreur << endl;
         cin.clear();
      }
      // on vide le buffer
      cin.ignore(numeric_limits<streamsize>::max(), '\n');
   } while(erreur);

   return saisie;
}

bool recommencer(const string& message, const char& option_vrai, const char& option_faux) {

   char choix;

   do {
      cout << message << "[" << option_vrai << "-" << option_faux << "]";
      cin >> choix;
   } while (   tolower(choix) != tolower(option_vrai)
            && tolower(choix) != tolower(option_faux));
```

```
   return tolower(choix) == tolower(option_vrai);
}

/*
-----------------------------------------------------------------------------
Nom du fichier : aleatoire.cpp
Auteur(s)      : Oscar Baume & Dorian Gillioz
Date creation  : 10.11.2021
Description    : Déclaration de la fonction de la librairie aleatoire
Remarque(s)    : Bibliothèque réutilisé du labo 05 Reflex
Modification   : ---
Compilateur    : Mingw-w64 g++ 11.2.0
-----------------------------------------------------------------------------
*/

#ifndef CPP_ALEATOIRE_H
#define CPP_ALEATOIRE_H

#include <cstdlib>
#include "preferences.h"

// Nom           : aleatoire
// Description   : Permet de donner un nombre aléatoire compris entre deux valeurs données
// Param min     : La valeur la plus petite de l'intervalle
// Param max     : La valeur la plus grande de l'intervalle
// Retour        : le nombre aléatoire
// Exception     : n/a
int aleatoire(int min, int max);

#endif //CPP_ALEATOIRE_H

/*
-----------------------------------------------------------------------------
Nom du fichier : aleatoire.cpp
Auteur(s)      : Oscar Baume & Dorian Gillioz
Date creation  : 10.11.2021
Description    : Définition de la fonction de la librairie aleatoire
Remarque(s)    : Bibliothèque réutilisé du labo 05 Reflex
Modification   : ---
Compilateur    : Mingw-w64 g++ 11.2.0
-----------------------------------------------------------------------------
*/

#include <cstdlib>
#include <time.h>
#include "Aleatoire.h"

using namespace std;

int aleatoire(int min, int max){
   static bool premiere_fois = true;

   // Si la fonction est appelée pour la première fois
   if (premiere_fois){
      // génere un seed pour les randoms
      srand((unsigned int)time(NULL));
      premiere_fois = false;
   }
   // retourne un nombre entre min et max
   return (int(unsigned(rand()) % (unsigned)(max - min + 1) + (unsigned)min));
}
```