Centre Informatique pour les Lettres et les Sciences Humaines

TD 15 : Les chameaux sur la dune

1 - Règles du jeu 2

2 - Cahier des charges et réflexions préalables 2

Distinction entre "représentation interne" et "présentation à l'écran" 3
Un dialogue "à géométrie variable" 3

3 - Création du projet et dessin de l'interface 3

4 - La fonction `main()` 4

5 - La classe `TD15DialogImpl` 5

Le constructeur 5
La fonction `dessineDune()` 6
La fonction `f_aide()` 7
Les fonctions `f_off()` et `f_on()` 7
La fonction `f_nouvellePartie()` 8
La fonction `f_deplace()` 8
Définition de la classe 9

6 - La classe `CDune` 9

Le constructeur 9
La fonction `deplace()` 10
La fonction `trouveLeBonCoup()` 10
Définition de la classe `CDune` 11

Le programme réalisé au cours du TD 15 permet à son utilisateur de jouer avec un petit casse-tête connu sous le nom du "problème des chameaux". Comme le montre l'image ci-contre, ce programme se singularise par la présence d'un menu et d'une fonction d'aide.

L'interface utilisateur du programme réalisé au cours du TD 15

1 - Règles du jeu

Deux caravanes composées d'un même nombre de chameaux (3, par exemple) se rencontrent sur la crête d'une dune. Le passage est trop étroit pour permettre le croisement normal, et les chameaux vont devoir sauter les uns par-dessus les autres pour pouvoir continuer leur chemin. Au début du problème, un espace libre sépare les deux caravanes :

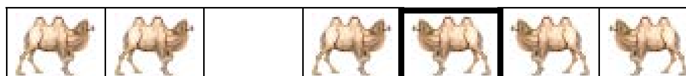
Le jeu consiste à découvrir la suite de déplacements qui permet d'obtenir l'état souhaité :

Deux types de mouvements permettent de modifier la position des chameaux :

1) Lorsqu'un chameau a l'espace vide immédiatement devant lui, il peut y avancer. A partir de la position initiale, deux mouvements de ce type sont possibles : avancer vers la droite

et avancer vers la gauche

2) Lorsqu'un chameau n'est séparé de l'espace vide que par le chameau qui est face à lui, il peut sauter par-dessus celui-ci pour aller dans la case vide. Ceci permet, par exemple, de passer de



à

On peut aussi autoriser un chameau à sauter par-dessus un membre de sa propre caravane, mais cela ne présente pas réellement d'intérêt, puisque cela conduit à la même position que si on les fait simplement avancer tous les deux. En tous cas, aucun de ces mouvements ne permet à un chameau de faire marche arrière.

2 - Cahier des charges et réflexions préalables

La fonction essentielle du programme est de permettre à l'utilisateur de jouer : il faut donc mettre en place la position initiale (avec, dans chaque caravane, le nombre de chameaux requis), puis permettre à l'utilisateur d'effectuer les mouvements légaux (et seulement ceux-ci). Le programme doit aussi être capable de fournir à l'utilisateur un conseil, c'est à dire de lui indiquer quel mouvement est susceptible de le conduire à la victoire à partir de la position courante.

Les mouvements sont effectués en cliquant sur le chameau qui doit aller dans la case vide. Si le déplacement tenté est interdit par les règles du jeu, un message doit l'indiquer. Lorsque la position finale est atteinte, un message de félicitation doit être affiché. Le bouton [Aide] fait clignoter le chameau dont le déplacement est conseillé. Le menu "Nouvelle partie" permet de choisir la taille des caravanes qui seront proposées.

Distinction entre "représentation interne" et "présentation à l'écran"

L'état actuel de la partie doit, de toute évidence, être présenté visuellement à l'écran. C'est sur cette présentation que l'utilisateur appuiera sa réflexion, et c'est en agissant sur ses éléments qu'il indiquera les mouvements qu'il souhaite effectuer.

Le programme a, lui aussi, besoin d'avoir accès à une représentation de la partie, notamment pour déterminer si le mouvement choisi par l'utilisateur est conforme aux règles du jeu. Il pourrait sembler logique d'écrire le programme de façon à ce qu'il utilise, pour ses calculs, la représentation de la partie proposée à l'utilisateur.

> Le programme pourrait, par exemple, vérifier la nature de l'image cliquée pour s'assurer que l'utilisateur n'essaie pas de faire reculer un chameau.

Cette façon de procéder respecterait un précepte important : nous évitons, habituellement, de représenter deux fois la même information, car toute modification doit alors être effectuée deux fois, et le risque apparaît qu'une erreur mette le programme en situation d'incohérence (deux valeurs différentes pour une même propriété, par exemple).

En l'occurrence, il nous faut cependant obéir à un précepte plus fondamental, que nous avons déjà rencontré lors du TD 10 (Bataille navale) :

La gestion de l'interface utilisateur et le traitement du problème doivent être pris en charge par des parties du programme aussi indépendantes l'une de l'autre que possible.

Ou, en d'autres termes :

Lorsqu'un programme effectue des traitements sur des données, il est généralement préférable qu'il dispose d'une représentation de celles-ci différente de celle proposée à l'utilisateur.

Ce précepte ne prend tout son poids que dans la mesure ou le programme effectue réellement des traitements. Lorsqu'il ne fait que prendre acte des actions effectuées par l'utilisateur, comme dans les TD 5 (Tic tac Toe), 8 (La patience des pharaons) ou 12 (La course d'escargots), il n'est pas très utile de le doter de sa propre représentation des données. S'il ne s'agissait ici que de permettre à l'utilisateur de déplacer les chameaux à sa guise (dans la limite des règles), nous pourrions sans doute nous passer d'une "représentation interne" de l'état de la partie. Mais le cahier des charges exige une fonction d'aide, et le programme va donc devoir se lancer dans des calculs dont seuls les résultats seront montrés à l'utilisateur.

> Dans les programmes qui distinguent présentation (à l'écran) et représentation (interne) du problème, il est souvent préférable de limiter le risque d'incohérence en créant une fonction qui rend l'affichage conforme à la représentation interne et en s'interdisant de modifier l'affichage autrement qu'en appelant cette fonction.

Un dialogue "à géométrie variable"

Le cahier des charges mentionne la possibilité de faire varier la taille des caravanes. Si nous choisissons d'utiliser des boutons pour représenter les différentes positions utilisables, ceci signifie que le dialogue affiché n'aura pas toujours le même nombre de boutons.

> Il serait, certes, possible de créer un dialogue ayant un nombre de boutons correspondant à la taille maximum autorisée pour les caravanes, et de se contenter de cacher les boutons inutiles lorsqu'une taille inférieure est utilisée. Procéder ainsi manquerait singulièrement d'élégance. Quant à l'idée de dessiner autant de dialogues qu'il y a de tailles de caravanes envisageables, mieux vaut ne rien en dire.

Nous allons donc utiliser un dialogue que son constructeur garni de widgets plutôt qu'un dialogue dont les widgets sont positionnés avec Qt Designer. Il nous sera donc possible d'adapter la taille des boutons (et des images représentant les chameaux) en fonction de la taille de caravanes choisie. Nous allons toutefois utiliser Qt Designer pour fixer quelques caractéristiques générales du dialogue et, surtout, pour créer les slots nécessaires.

3 - Création du projet et dessin de l'interface

Utilisez la procédure habituelle (cf. aide mémoire) pour créer un projet nommé TD15 ☐ et effacez tous les widgets présents dans le dialogue ☐.

Donnez à la propriété "Font – Point size" du dialogue la valeur 14 ☐.

Créez (Menu "Edit", choix "Slots...") les slots suggérés ci-contre ☐.

Function	Return Type	Specifier	Access	Type	In Use
f_aide()	void	virtual	public	slot	No
f_deplace(int)	void	virtual	public	slot	No
f_nouvellePartie(int)	void	virtual	public	slot	No
f_off()	void	virtual	public	slot	No
f_on()	void	virtual	public	slot	No

Un clic sur le bouton [Au secours !] appellera la fonction `f_aide()`

Un clic dans le buttonGroup contenant les chameaux appellera `f_deplace(int)`

La sélection d'un item du menu "Nouvelle partie" appellera `f_nouvellePartie(int)`

Les slots `f_off()` et `f_on()` seront, pour leur part, utilisés par les timers assurant le clignotement du bouton correspondant au coup suggéré par `f_aide()`.

Enregistrez votre dialogue et retournez dans Visual C++ ☐.

4 - La fonction `main()`

Lorsqu'on ajoute dynamiquement un widget à un dialogue (cf. TD 12 et 13), on confie à celui-ci la responsabilité de la destruction du widget en question, et cette destruction n'aura lieu que lors de la destruction du dialogue. Nous ne pouvons donc pas supprimer les widgets présents dans un dialogue pour les remplacer par d'autres : il nous faut, en fait, supprimer purement et simplement le dialogue et en créer un nouveau de toutes pièces. L'instanciation de la classe de dialogue étant du ressort de la fonction `main()`, c'est dans celle-ci que nous allons mettre en place la logique qui va permettre à l'utilisateur de changer la taille des caravanes :

```
1  int main( int argc, char** argv )
2  {
3  QApplication app(argc, argv);
4  int reponse;
5  int taille = 3;
6  do {
7     TD15DialogImpl dialog(taille, 0, 0, TRUE );
8     app.setMainWidget(&dialog);
9     reponse = dialog.exec();
10    taille = dialog.tailleSouhaitee();
11    } while (reponse == QDialog::Accepted);
12 return 0;
13 }
```

Deux variables locales permettent respectivement de stocker (9) la valeur renvoyée par `exec()`et de communiquer (7) au constructeur du dialogue la longueur des caravanes devant être représentées.

1 : Le constructeur de `TD15DialogImpl` devra accepter un premier argument de type `int`.

L'exécution de la ligne 9 ne s'achève que lorsqu'une action de l'utilisateur provoque la fermeture du dialogue qui lui est proposé. Deux cas peuvent alors se présenter :

- Soit l'utilisateur a cliqué sur la case de fermeture pour mettre fin au programme. La fonction `exec()` renvoie alors la valeur `QDialog::Rejected`, ce qui provoquera la fin de la boucle et, donc, de la fonction `main()`, c'est à dire celle du programme lui-même.
- Soit l'une des fonctions membre de la classe `TD15DialogImpl` a appelé `accept()`. La fonction `exec()` renvoie alors la valeur `QDialog:: Accepted`, ce qui provoquera un nouveau passage dans la boucle avec création d'un nouveau dialogue.

Remarquez que les caravanes de ce nouveau dialogue auront une taille qui dépend de la valeur renvoyée (10) par la fonction `tailleSouhaitee()` : c'est en effet le dialogue qui vient de se fermer qui avait recueilli cette information, et il faut la transférer dans une variable locale avant que la fin du bloc (11) ne se traduise par la destruction du dialogue en question !

2 : La classe `TD15DialogImpl` devra comporter une fonction nommée `tailleSouhaitee()` renvoyant le choix de l'utilisateur concernant la taille des caravanes de sa prochaine partie.

5 - La classe TD15DialogImpl

Notre classe de dialogue se singularise par un constructeur inhabituellement long. Cette longueur est rendue acceptable par la simplicité structurelle de la fonction, qui ne comporte que très peu de boucles et de tests, jamais enchâssés les uns dans les autres. Le code se laisse par ailleurs facilement segmenter en paragraphes relativement indépendants les uns des autres.

Le constructeur

Pour répondre à la première promesse faite par la fonction `main()`, le constructeur est doté d'un premier paramètre permettant de lui communiquer la taille des caravanes que devra gérer le dialogue en cours de création. La valeur reçue est utilisée dans la liste d'initialisation, pour initialiser la variable membre qui contiendra la représentation interne de l'état de la partie.

3 : La classe `CDune` est utilisée pour représenter l'état de la partie. Elle doit comporter un constructeur qui accepte un argument unique indiquant la taille des caravanes.

```
1  TD15DialogImpl::TD15DialogImpl(int taille, QWidget* parent, const char* name,
       bool modal, WFlags f ) : TD15Dialog(parent, name, modal, f ), m_dune(taille)
2  {
   //mise en place des timers de clignotement (cf. TD12)
3  m_timerOn = new QTimer(this);
4  connect(m_timerOn, SIGNAL(timeout()), this, SLOT(f_on()) );
5  m_timerOff = new QTimer(this);
6  connect(m_timerOff, SIGNAL(timeout()), this, SLOT(f_off()) );
```

Il faut ensuite préparer les images représentant les chameaux et la case vide. Ces images doivent être accessibles aux autres fonctions membre et sont donc stockées dans deux variables membre : `m_cham` et `m_caseVide`. On suppose qu'un fichier chameau.png se trouve dans le même dossier que le programme en cours d'exécution (cf. aide-mémoire). La largeur maximale acceptable pour l'image dépend du nombre de cases devant être gérées et de l'ordre de grandeur du dialogue souhaité. Si l'image fournie s'avère trop grande, un ajustement de taille est nécessaire, et il faut veiller à diminuer la hauteur dans les même proportions pour éviter de déformer l'image :

```
   //création des pixmap
7  QFileInfo info(*qApp->argv());
8  m_cham.load(info.dirPath() + "/chameau.png");
9  double largeur = (800-6*m_dune.taille()) / m_dune.taille();
10 if(largeur < m_cham.width())
11    m_cham = m_cham.smoothScale(largeur, (largeur * m_cham.height()) / m_cham.width() );
12 m_caseVide = m_cham.copy();   //pour lui donner la bonne taille
13 m_caseVide.fill(m_cham.pixel(0,0));
```

La case vide sera, pour sa part, représentée par une image de même taille que celle représentant un chameau *(12)*, uniformément remplie avec la couleur de fond de celle-ci *(13)*.

4 : Il existe une fonction `CDune::taille()` qui renvoie le nombre de cases nécessaires au jeu.

L'étape suivante est la mise en place du menu : celui-ci est créé *(14)*, puis connecté à la fonction qui le prendra en charge *(15)*. Les différents items qu'il doit proposer (les valeurs 1, 2, 3, 4, 5, 6 et 7, en l'occurrence) lui sont ensuite ajoutés *(16-18)*. Le menu est finalement ajouté *(20)* à la barre de menu créée *(19)* pour le recevoir.

```
   //création du menu
14 QPopupMenu * leMenu = new QPopupMenu(this);
15 connect(leMenu, SIGNAL(activated(int)), this, SLOT(f_nouvellePartie(int)) );
16 int n;
17 for(n=1 ; n < 8 ; ++n)
18    leMenu->insertItem(QString::number(n), n);
19 QMenuBar *laBarre = new QMenuBar(this);
20 laBarre->insertItem("&Nouvelle partie", leMenu );
```

Lors de la création d'un item de menu, deux valeurs sont passées : le texte qui représentera l'item dans le menu et la valeur qui sera transmise à la fonction connectée au menu lorsque

l'item sera sélectionné. Dans le cas présent la première valeur est simplement la représentation écrite (numération positionnelle en base 10, chiffres arabes) de la seconde.

Il faut ensuite créer le groupe de boutons figurant les cases du jeu. Ces boutons sont affectés (28) à un buttonGroup, ce qui leur permet d'être connectés (22) à une même fonction. Ils sont également placés (29) dans un layout horizontal qui assure leur alignement correct.

```
   //la dune
21 QButtonGroup * groupeBoutons = new QButtonGroup(this);
22 connect(groupeBoutons, SIGNAL(clicked(int)), this, SLOT(f_deplace(int)) );
23 groupeBoutons->setLineWidth(0); //il n'a aucun rôle visuel
24 QHBoxLayout * boxChameaux = new QHBoxLayout();
25 for (n=0 ; n < m_dune.taille() ; ++n)
26    {
27    m_lesBoutons[n] = new QPushButton(this);
28    groupeBoutons->insert(m_lesBoutons[n], n);
29    boxChameaux->addWidget(m_lesBoutons[n]);
30    }
```

Les adresses des boutons sont placés dans une QMap qui permettra aux fonctions membre qui en ont besoin de modifier l'image affichée par chacun d'entre eux.

Le dernier élément de l'interface est le bouton d'aide. Le layout où il est placé contient, à sa gauche et à sa droite (36 et 38), des "écarteurs" l'empêchant d'occuper toute la largeur du dialogue.

```
   //le bouton d'aide
31 QHBoxLayout * boxBouton = new QHBoxLayout();
32 b_aide = new QPushButton("Au secours !",this);
33 connect(b_aide, SIGNAL(clicked()), this, SLOT(f_aide()) );
34 b_aide->setDefault(true);
35 b_aide->setFocus();
36 boxBouton->addStretch(5);
37 boxBouton->addWidget(b_aide);
38 boxBouton->addStretch(5);
```

L'adresse du bouton d'aide est, elle aussi, placée dans une variable membre, ce qui permettra de désactiver cette fonction lorsque la partie sera terminée.

Une fois tous les éléments construits, il ne reste plus qu'à les assembler pour former le dialogue :

```
   //mise en place finale
39 QVBoxLayout * general = new QVBoxLayout(this);
40 general->addLayout(boxChameaux);
41 general->addLayout(boxBouton);
42 dessineDune();
43 }
```

Donnez au corps de votre constructeur le contenu suggéré ci-dessus ☐.

La fonction `dessineDune()`

Le rôle de cette fonction est de placer dans chacun des boutons figurant les cases du jeu l'image adéquate pour représenter l'état actuel de la partie. Comme nous le savons, cet état est représenté dans la variable `m_dune`, et c'est tout naturellement à cette variable que la fonction `dessineDune()` demande ce qu'il faut dessiner dans chaque case :

```
1  void TD15DialogImpl::dessineDune()
2  {
3  assert (m_dune.taille() == m_lesBoutons.count());
4  int n;
5  for(n=0 ; n < m_dune.taille() ; ++n)
6     switch(m_dune[n])            5
7        {
8        case CHAM_G : m_lesBoutons[n]->setPixmap(m_cham); break;
9        case TROU : m_lesBoutons[n]->setPixmap(m_caseVide); break;
10       case CHAM_D : m_lesBoutons[n]->setPixmap(m_cham.mirror(true, false)); break;
11       }
12 }
```

5 : La classe CDune dispose d'un opérateur [] qui permet de connaître le contenu d'une case. Cet opérateur renvoie l'une des valeurs du type énuméré EContenuCase: CHAM_G, CHAM_D ou TROU.

Comme la fonction dessineDune() dépend totalement du fait que le dialogue dispose exactement du nombre de boutons permettant de représenter le jeu, elle comporte (3) une assertion (cf. aide-mémoire) qui, d'une part, arrêtera l'exécution si une erreur a conduit le programme à violer cette condition et, d'autre part (et c'est peut-être le plus important...), rappelle à tout lecteur du code source qu'il s'agit là d'un présupposé essentiel du programme.

Le dessin d'un chameau "qui va dans l'autre sens" est obtenu, à partir de l'image "normale", par symétrie sur l'axe horizontal (mais pas sur l'axe vertical).

Ajoutez une fonction dessineDune() à votre dialogue et donnez-lui le contenu décrit ci-dessus ☐.

La fonction f_aide()

Pour déterminer quel chameau doit être déplacé, la fonction f_aide() se contente de poser la question à m_dune. Cette valeur est stockée dans une variable membre, ce qui la rend disponible pour les fonctions assurant le clignotement du bouton correspondant.

6 : La fonction CDune::trouveLeBonCoup() renvoie le numéro de la case dont le chameau doit être déplacé pour que la partie puisse être gagnée. En cas d'impossibilité, elle renvoie –1.

```
1 void TD15DialogImpl::f_aide()
2 {
3 m_conseil = m_dune.trouveLeBonCoup();
4 if(m_conseil < 0)
5   QMessageBox::critical(this, "Désolé...","Il n'est plus possible de gagner.");
6 f_off();
7 }
```

La fonction f_aide() s'achève par un appel à la fonction qui va faire disparaître le chameau dont le déplacement est conseillé.

Ajoutez une fonction f_aide() à votre dialogue et donnez-lui le contenu suggéré ci-dessus ☐.

Les fonctions f_off() et f_on()

Si le conseil est utilisable, f_off() remplace l'image contenue dans le bouton correspondant par celle de la case vide, puis programme le timer qui va appeler la fonction f_on().

```
1 void TD15DialogImpl::f_off()
2 {
3 if(m_conseil < 0)
4   return;
5 m_lesBoutons[m_conseil]->setPixmap(m_caseVide);
6 m_timerOn->start(150, true);
7 }
```

Remarquez l'usage du second paramètre de la fonction start() : le timerOn provoquera un unique événement timeout() et s'arrêtera de lui-même.

La fonction f_on() rétablit l'apparence normale du jeu, et programme un appel de la fonction f_off(), ce qui assurera le clignotement du chameau conseillé, jusqu'à ce que la valeur contenue dans m_conseil devienne négative :

```
1 void TD15DialogImpl::f_on()
2 {
3 dessineDune();
4 m_timerOff->start(350, true);
5 }
```

Ajoutez les fonctions f_off()et f_on() à votre dialogue et donnez-leur les contenus décrits ci-dessus ☐.

La fonction f_nouvellePartie()

Cette fonction reçoit un argument qui indique quel item du menu qui lui est connecté a été sélectionné par l'utilisateur. Dans le cas présent, cette valeur correspond directement à la taille des caravanes qui doivent être proposées dans la prochaine partie, et la fonction se contente donc d'attribuer à m_dune une valeur correspondant à l'état initial d'une telle partie.

Si la taille des caravanes de la prochaine partie diffère de celles de la partie en cours, il faut reconstruire le dialogue, et la fonction f_nouvellePartie() met donc fin à l'existence de l'instance de TD15DialogImpl en cours d'activité en appelant accept(), ce qui signale à la fonction main() qu'elle doit réinstancier TD15DialogImpl pour proposer une nouvelle partie.

```
1 void TD15DialogImpl::f_nouvellePartie(int tailleCaravane)
2 {
3 m_dune = CDune(tailleCaravane);
4 if(m_dune.taille() != m_lesBoutons.count())
5   accept();       //main() va créer un nouveau dialogue
6 else
7   dessineDune(); //on continue avec le dialogue actuel
8 }
```

Ajoutez une fonction f_nouvellePartie() à votre dialogue et donnez-lui le contenu décrit ci-dessus ☐.

La fonction f_deplace()

La fonction f_deplace() ne doit pas chercher à vérifier elle-même la légalité du déplacement du chameau sur lequel l'utilisateur a cliqué, et encore moins à déterminer les conséquences du mouvement (lorsque celui-ci est possible). Elle doit au contraire déléguer cette tâche à une fonction membre de CDune et se contenter de mettre à jour l'affichage si le déplacement est accepté (15 ou 20) :

```
1  void TD15DialogImpl::f_deplace(int qui)
2  {
3  m_conseil = -1; //arrête l'aide
4  switch(m_dune.deplace(qui))
5    {
6    case TROP_LOIN :
7      QMessageBox::critical(this, "Mouvement impossible :",
                                              "La case vide est trop loin !");
8      break;
9    case MAUVAIS_SENS :
10     QMessageBox::critical(this, "Mouvement impossible :",
                                                  "Pas de marche arrière !");
11     break;
12   case DEJA_VIDE :
13     break ;
14   case VICTOIRE :
15     dessineDune();
16     b_aide->setEnabled(FALSE);
17     QMessageBox::warning(this, "Bravo :", "Vous avez gagné !");
18     break;
19   case NORMAL :
20     dessineDune();
21   }
22 }
```

6 Les cinq cas de figure que la fonction CDune::deplace() peut rencontrer sont décrits par un type énuméré qui confère à f_deplace() une lisibilité maximale.

7 : La fonction CDune::deplace() renvoie une valeur du type énuméré EResultatMvmt qui décrit les conséquences du coup qui lui a été transmis comme argument.

Ajoutez une fonction f_deplace() à votre dialogue et donnez-lui le contenu décrit ci-dessus ☐.

Définition de la classe

Une fois munie des variables membre nécessaires (5-12) et de la fonction (21) qui permet à `main()` de déterminer la taille du dialogue à créer, la classe `TD15DialogImpl` prend cette forme :

```
1  class TD15DialogImpl : public TD15Dialog
2  {
3     Q_OBJECT
4  protected:
5     QMap<int, QPushButton *> m_lesBoutons;
6     CDune m_dune;
7     QImage m_cham;
8     QImage m_caseVide;
9     int m_conseil;
10    QTimer * m_timerOn;
11    QTimer * m_timerOff;
12    QPushButton *b_aide;
      //fonctions membre
13    void dessineDune();
14    void f_off();
15    void f_on();
16    void f_aide();
17    void f_nouvellePartie(int tailleCaravane);
18    void f_deplace(int position);
19 public:
20    TD15DialogImpl(int taille, QWidget* parent = 0, const char* name = 0,
                                              bool modal = FALSE, WFlags f = 0 );
21    int tailleSouhaitee() const {return m_dune.taille() / 2;}
22 };
```

Faites en sorte que votre fichier td15dialogimpl.h contienne un code équivalent à celui décrit ci-dessus ☐ et essayez d'y ajouter les directives `#include` nécessaires ☐.

6 - La classe CDune

La classe `CDune` doit réaliser les cinq promesses qui ont été faites mais n'ont pas encore été tenues :

	Origine	Nature de la promesse
3	`TD15DialogImpl()`	`CDune::CDune(int tailleCaravane)`
4	`TD15DialogImpl()`	`int CDune::taille()`
5	`TD15DialogImpl()`	`EContenuCase CDune::operator [] (int)`
6	`f_aide()`	`int CDune::trouveLeBonCoup()`
7	`f_deplace()`	`EResultatMvmt CDune::deplace(int)`

Le constructeur

La création d'une `CDune` doit prendre en charge la représentation d'une partie dans son état initial. L'état d'une partie peut être représenté de différentes façons, la plus logique étant peut-être l'utilisation d'une `QTable<int, EContenuCase>`. Une simple `QString` permet cependant de simplifier un peu la création de l'état initial et la détection de la victoire :

```
1  CDune::CDune(int t) : m_posTrou(t)
2  {
3  m_etat = QString().fill(CHAM_G, t) + QChar(TROU) + QString().fill(CHAM_D, t);
4  m_victoire = QString().fill(CHAM_D, t) + QChar(TROU) + QString().fill(CHAM_G, t);
5  }
```

Deux variables membre de type `QString` sont utilisées pour représenter l'état actuel et l'état souhaité. Une troisième variable membre, de type `int`, contient le numéro de la case actuellement vide, ce qui évitera d'avoir à scruter l'état de la partie chaque fois que cette information est nécessaire.

Créez la classe `CDune` et donnez au corps de son constructeur le contenu suggéré ci-dessus ☐.

La fonction deplace()

La définition de la fonction `f_deplace()` ne pose aucun problème particulier :

```
1  EResultatMvmt CDune::f_deplace(int pos)
2  {
   //vérifie que le coup est légal
3  if(pos == m_posTrou)
4     return DEJA_VIDE;
5  if(abs(pos-m_posTrou) > 2)
6     return TROP_LOIN;
7  if(pos > m_posTrou && m_etat[pos] == CHAM_G)
8     return MAUVAIS_SENS;
9  if(pos < m_posTrou && m_etat[pos] == CHAM_D)
10    return MAUVAIS_SENS;
   //effectue le mouvement
11 m_etat[m_posTrou] = m_etat[pos];
12 m_etat[pos] = TROU;
13 m_posTrou = pos;
   //annonce le résultat
14 return (m_etat == m_victoire) ? VICTOIRE : NORMAL;
15 }
```

Remarquez quand-même l'utilisation d'une fonction de la librairie standard permettant d'obtenir la valeur absolue d'un nombre et l'accès *(11-12)* aux caractères de la `QString m_etat` à l'aide de l'opérateur `[]`.

Ajoutez une fonction `f_deplace()` à votre classe `CDune` et donnez-lui le contenu suggéré ci-dessus ☐.

La fonction trouveLeBonCoup()

Cette fonction est incontestablement la plus intéressante du programme, d'une part parce qu'elle ne se contente pas de gérer les actions de l'utilisateur mais effectue un véritable traitement, et d'autre part parce qu'elle se révèle étonnamment brève :

```
1  int CDune::trouveLeBonCoup()
2  {
3  int coupEnvisage;
4  for(coupEnvisage = 0 ; coupEnvisage < taille() ; ++ coupEnvisage)
5     {
6     CDune hypothese (*this);
7     switch(hypothese.f_deplace(coupEnvisage))
8        {
9        default :
10          break; //le coupEnvisage est illégal…
11       case VICTOIRE :
12          return coupEnvisage;
13       case NORMAL :
14          if (hypothese.trouveLeBonCoup() >= 0)
15             return coupEnvisage;
16       }
17    }
18 return -1;
19 }
```

Pour trouver le bon coup, cette fonction entreprend l'examen de tous les déplacements envisageables. Si aucun de ces déplacements ne permet de gagner la partie *(18)*, la fonction renvoie –1, conformément à la promesse n°6 (page 7).

Pour examiner un déplacement, la fonction `trouveLeBonCoup()` commence par créer une copie de la représentation de l'état actuel de la partie, puis elle essaie d'effectuer le déplacement sur cette copie. Trois cas peuvent alors se présenter :

1) Le `coupEnvisage` peut être interdit par les règles. La fonction `deplace()` renvoie alors `TROP_LOIN`, `MAUVAIS_SENS` ou `DEJA_VIDE` et `trouveLeBonCoup()` n'a rien de spécial à faire *(10)*, il lui faut simplement continuer à examiner l'ensemble des hypothèses envisageables.

La présence des lignes *9* et *10* ne change strictement rien au comportement du programme. Elle ne vise qu'à expliciter le fait qu'aucun traitement n'est effectué dans ces cas.

2) Le `coupEnvisage` peut produire la position finale visée. C'est donc incontestablement "le bon coup" et il suffit de le renvoyer *(12)* à celui qui a posé la question.

3) Le coup envisagé peut être légal mais conduire à une position qui n'est pas la position recherchée (la fonction `deplace()` renvoie alors `NORMAL`). Pour savoir si cette position est souhaitable, il faut savoir si, à partir d'elle, une séquence de coups existe qui conduit à la victoire.

Si la position s'avère souhaitable, le coup qui y conduit est le bon, il faut donc le renvoyer *(15)*. Dans le cas contraire, il faut continuer à examiner l'ensemble des hypothèses envisageables.

Comment savoir si la position produite par le `coupEnvisage` est souhaitable ? Il suffit de le lui demander ! En effet, si "à partir d'elle, une séquence de coups existe qui conduit à la victoire", cela signifie que, appelée à son titre, la fonction `trouveLeBonCoup()` renverra un coup préconisé et non la valeur –1 qui indique que la situation est sans espoir.

Ajoutez une fonction `f_trouveLeBonCoup()` à votre classe `CDune` et donnez-lui le contenu suggéré ci-dessus ☐.

Définition de la classe CDune

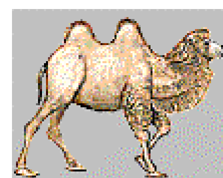
Il reste à tenir les promesses 4 et 5 en munissant la classe `CDune` des fonctions qui permettent d'obtenir la taille du plateau de jeu géré par une instance et le contenu d'une des cases de ce plateau. Le contenu du fichier cdune.h se présente donc finalement ainsi :

```
1  enum EContenuCase {CHAM_G , TROU, CHAM_D};
2  enum EResultatMvmt {NORMAL, DEJA_VIDE, TROP_LOIN, MAUVAIS_SENS, VICTOIRE};
3  class CDune
4  {
5  public:
6     CDune(int t);
7     int taille() const {return m_etat.length();}
8     char operator[] (int p) const {return m_etat[p];}
9     int trouveLeBonCoup();
10    EResultatMvmt deplace(int qui);
11 protected:
12    QString m_etat;
13    QString m_victoire;
14    int m_posTrou;
15 };
```

Vous pouvez aussi préférer faire figurer les définitions des types énumérés dans leur propre fichier .h, que vous inclurez alors là où ces définitions sont nécessaires.

Faites en sorte que votre fichier cdune.h contienne un code équivalent à celui décrit ci-dessus ☐ et essayez d'y ajouter les directives `#include` nécessaires ☐.

Enregistrez ce fichier dans le dossier contenant votre exécutable (le dossier debug de votre projet, ou tout autre dossier où vous souhaitez installer le programme) sous le nom chameau.png (attention, le fichier disponible sur le web n'a pas la bonne extension, ce qui évite à votre navigateur de l'ouvrir au lieu de vous proposer de le sauvegarder…)



Si le lien vers le fichier ne fonctionne pas, vous pouvez copier-coller dans Paint l'image ci-contre et "enregistrer sous" au format png.

Vous pouvez maintenant compiler votre programme ☐ et, si vous n'avez oublié aucune directive `#include`, apprendre à résoudre le problème des chameaux sur la dune ☐.