



C++_PROJECT_MOROCCAN CARDS GAME



Qt

Réalise par les étudiants :

Ouriaghli Touil Soufian
Cheairi Hamza

FSTT : 2023/2024

Rapport du jeu de carte en C++ et QT

Ce rapport présente les différentes parties du jeu réalisé comme projet du module (POO en C++).

Le jeu porte le nom (le DOS) créé en appliquant les connaissances et les compétences requissent en (POO en C++) enseigné par professeur IKRAM BEN ABDEL OUAHAB et en utilisant la documentation QT et quelques formations vidéo (sur YOUTUBE).

Afin de réaliser ce projet on a partagé les tâches moitié moitié et pour plus de compréhension et d'apprentissage notre binôme a décidé de travailler toutes les parties ensemble.

Avant de commencer à détailler les différentes parties du jeu on tient à remercier notre professeur de cours et notre encadrant de projet [prof IKRAM BEN ABDELOUAHAB](#)

Choix du jeu : le DOS

Le jeu du DOS est un jeu qui n'est pas assez populaire au Maroc mais en même temps un jeu stratégique : le joueur doit penser rapidement à une stratégie pour se débarrasser de ses cartes en fonction de la première carte posé au milieu de la table et en fonction des cartes joué par son adversaire.

Motive pour ce choix : les jeux de cartes RONDA et HEZZ-2 sont plus populaires on pensait sérieusement choisir HEZZ-2. on a commencé le

développement de HEZZ-2 mais à un certain moment on a su qu'il y a pas mal de binôme qui ont choisi HEZZ-2 d'où on a décidé de changer de jeu et choisir un jeu spécial pour nous on a essayé de travailler sur quatre copie mais c'était assez difficile de simplifier les règles du jeu pour que le jeu peut être joué entre deux joueurs à la place de quatre joueur , RONDA n'était pas un choix possible pour la même raison de HEZZ-2 à la fin on a arrivé à choisir le jeu DOS .

Les outils de développement

- **Les outils qt (qt Creator/qt designer)**

Qt Creator est un environnement de développement intégré (IDE) puissant conçu spécifiquement pour faciliter le développement d'applications utilisant le Framework Qt.

Qt Creator offre une gamme complète d'outils et de fonctionnalités visant à simplifier le processus de création d'applications multiplateformes, notamment des applications graphiques, des applications mobiles, des applications web et bien plus encore.

- **Langage de programmation C++**

Le langage de programmation C++ s'impose comme l'un des choix privilégiés des développeurs, notamment dans le domaine des applications. Il offre la possibilité de développer selon plusieurs

paradigmes, dont la programmation générique, procédurale et orientée objet.

Pour notre projet, nous avons choisi de nous appuyer sur le paradigme orienté objet en C++, une approche informatique qui consiste à définir et faire interagir des objets au moyen de différentes technologies.

Problèmes rencontrés lors du travail sur le projet

Au cours de la mise en œuvre de notre projet ambitieux, nous avons dû surmonter divers obstacles qui ont enrichi notre parcours de développement.

Ces défis, bien que parfois arduis, ont façonné notre expérience de manière inattendue et ont contribué à notre apprentissage.

Voici une expansion détaillée de chacun des problèmes rencontrés :

Difficulté à Trouver des Ressources en Ligne pour l'Apprentissage de Qt :

La première étape de notre périple a été marquée par la recherche incessante de ressources en ligne pour maîtriser l'environnement de développement Qt.

Malgré l'abondance d'informations disponibles, trier et sélectionner des ressources de qualité s'est avéré être une tâche exigeante.

La diversité des supports a ajouté une complexité supplémentaire, nécessitant une navigation minutieuse à travers une myriade de tutoriels, De forums et de documentations.

Complexité de l'Apprentissage des Widgets de Qt :

L'appréhension des widgets dans le cadre du Framework Qt s'est avérée être une entreprise laborieuse. Chaque widget a ses propres particularités et fonctionnalités, exigeant une immersion approfondie dans la documentation officielle et la réalisation de multiples exemples pratiques.

Cette phase d'apprentissage intensif a inévitablement rallongé notre calendrier de développement initial.

Problématique Liée à la Logique du Jeu :

La conceptualisation et la mise en œuvre de la logique du jeu ont constitué un défi intellectuel majeur.

Nous avons été confrontés à des dilemmes complexes, tels que la conception de mécanismes de jeu interactifs et la résolution de problèmes liés à la gestion de l'état du jeu.

Les multiples itérations nécessaires pour parvenir à une logique de jeu cohérente ont entraîné des retards imprévus dans notre échéancier.

Débogage du Code et Découragement :

En dépit de notre engagement inflexible, le processus de développement n'a pas été sans embûches.

Les bugs persistants dans le code ont parfois conduit à des moments de frustration et de découragement.

Le débogage rigoureux a été nécessaire, avec des cycles itératifs de correction de bugs, test et retest.

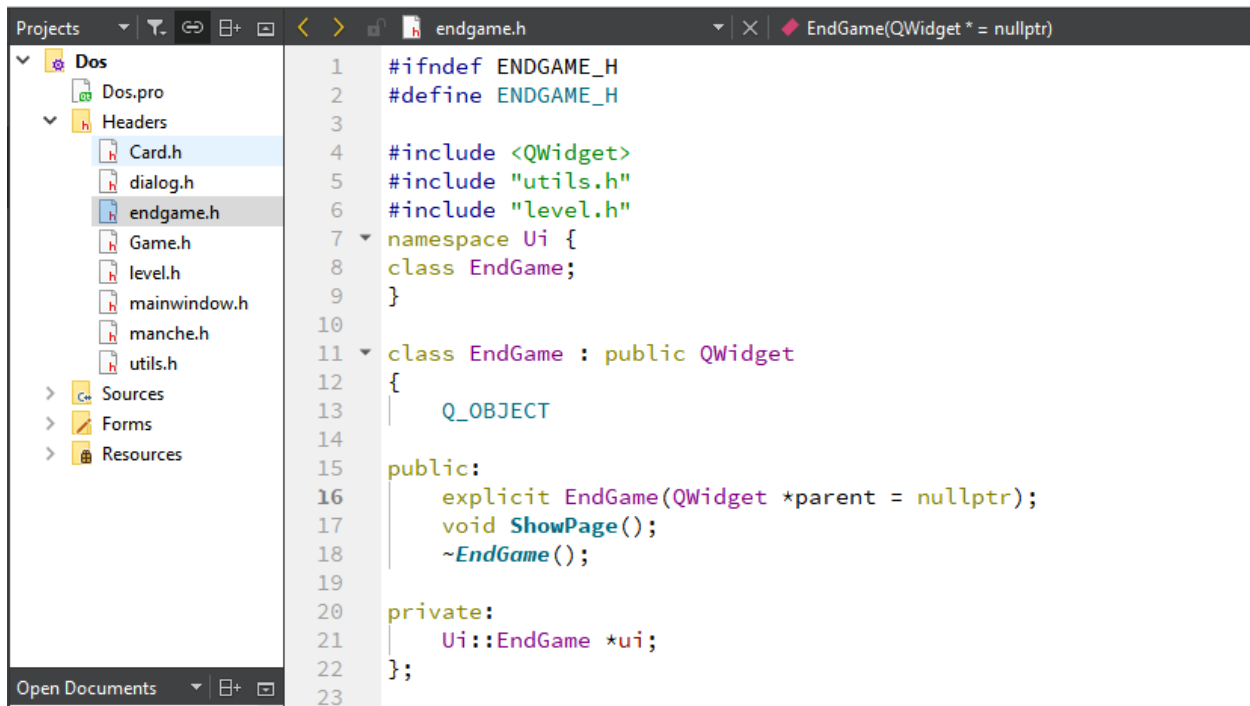
Ces défis imprévus ont rallongé notre effort global, mais ont également renforcé notre résilience face aux obstacles techniques. En conclusion, bien que ces défis aient semblé initialement décourageants, ils ont contribué de manière inattendue à notre croissance en tant que développeurs.

L'ensemble du processus, du combat initial pour trouver des ressources à la résolution minutieuse des bugs, a forgé une expérience complète, nous préparant ainsi à relever des défis plus complexes à l'avenir.

Différente classe du projet :

Dans notre implémentation du jeu, nous avons pris soin de spécifier chaque état et chaque composant du jeu au moyen de classes uniques, afin d'apporter une clarté maximale au code. Cette approche orientée objet nous a permis de modéliser de manière précise les différentes entités du jeu, comme les cartes, les joueurs, et les différentes phases de jeu. Chaque classe est conçue pour encapsuler les données et le comportement associés à une entité spécifique, facilitant ainsi la compréhension, la maintenance et l'extension du code. Cette organisation hiérarchique nous a également permis d'adopter une structure modulaire, favorisant la réutilisation du code et la gestion efficace des interactions entre les différentes parties du jeu. En détaillant chaque aspect du jeu à l'aide de classes dédiées, notre objectif est d'offrir une base solide pour le développement continu du jeu et de fournir une documentation exhaustive sur l'architecture du code source.

✓ Les différentes classes et « Headers » utilisées :



The screenshot shows the Qt Creator IDE with the 'endgame.h' file open. The left sidebar displays a project tree with 'Dos' as the root, containing 'Dos.pro', a 'Headers' folder with 'Card.h', 'dialog.h', 'endgame.h', 'Game.h', 'level.h', 'mainwindow.h', 'manche.h', and 'utils.h', and 'Sources' folders. The main editor shows the code for 'endgame.h'.

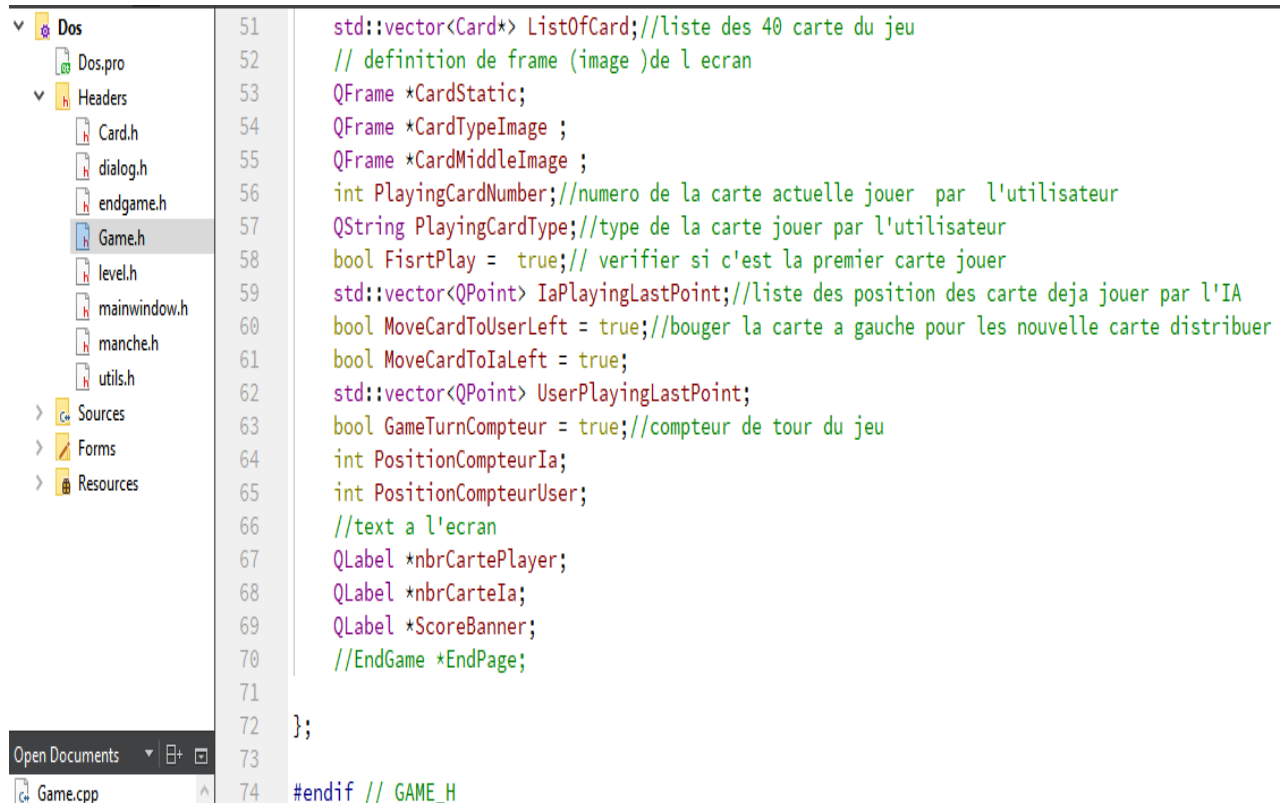
```
1  #ifndef ENDGAME_H
2  #define ENDGAME_H
3
4  #include <QWidget>
5  #include "utils.h"
6  #include "level.h"
7  namespace Ui {
8  class EndGame;
9  }
10
11 class EndGame : public QWidget
12 {
13     Q_OBJECT
14
15 public:
16     explicit EndGame(QWidget *parent = nullptr);
17     void ShowPage();
18     ~EndGame();
19
20 private:
21     Ui::EndGame *ui;
22 };
23
```

• Classe principale : Classe Game :



The screenshot shows the Qt Creator IDE with the 'Game.h' file open. The left sidebar displays a project tree with 'Dos' as the root, containing 'Dos.pro', a 'Headers' folder with 'Card.h', 'dialog.h', 'endgame.h', 'Game.h', 'level.h', 'mainwindow.h', 'manche.h', and 'utils.h', and 'Sources' folders. The main editor shows the code for 'Game.h'.

```
1  #ifndef GAME_H
2  #define GAME_H
3
4  #include "utils.h"
5  #include <QGraphicsScene>
6  #include <QGraphicsView>
7  #include <QPropertyAnimation>
8  #include <QPixmap>
9  #include <QMainWindow>
10 #include <Qpoint>
11 #include <QLabel>
12 #include <Card.h>
13 #include <QFrame>
14 #include "endgame.h"
15 #include "manche.h"
16
17 class Game : public QMainWindow
18 {
19     Q_OBJECT
20 public:
21     Game(QWidget *parent = nullptr);
22     void SetPosition(bool Top);
23     void FirstAnimation();
24     void InitCard();
25     void FirstAnimation(bool Direction);
26 }
```

✓ Analyse de cette classe :

Bibliothèques incluses :

Inclusion des bibliothèques nécessaires, telles que `utils.h`, les classes de `QGraphics` (pour la gestion graphique), `QPropertyAnimation` (pour les animations), `QPixmap` (pour les images), etc.

Déclaration de la classe Game :

La classe `Game` hérite de la classe `QMainWindow` du framework `Qt` et utilise le système de signaux et de slots (indiqué par `Q_OBJECT`).

Membres publics :

Méthodes publiques pour initialiser le jeu, définir la position, effectuer la première animation, gérer les joueurs (utilisateur et IA), effectuer des animations de carte, mélanger les cartes, vérifier les possibilités de jeu, effectuer des déplacements de carte, activer/désactiver des cartes, et gérer la logique du jeu.

Membres privés :

Attributs privés, notamment la hauteur et la largeur de l'écran, les positions des cartes pour les joueurs, les listes de cartes pour l'utilisateur, l'IA et toutes les cartes, des éléments graphiques comme des cadres et des labels, des compteurs pour suivre l'état du jeu, des listes de points pour les positions des cartes, etc.

Méthodes privées :

Méthodes privées pour l'animation des mouvements de cartes, la gestion des cartes jouées, l'affichage de l'écran de fin et de la manche.

Destructeur :

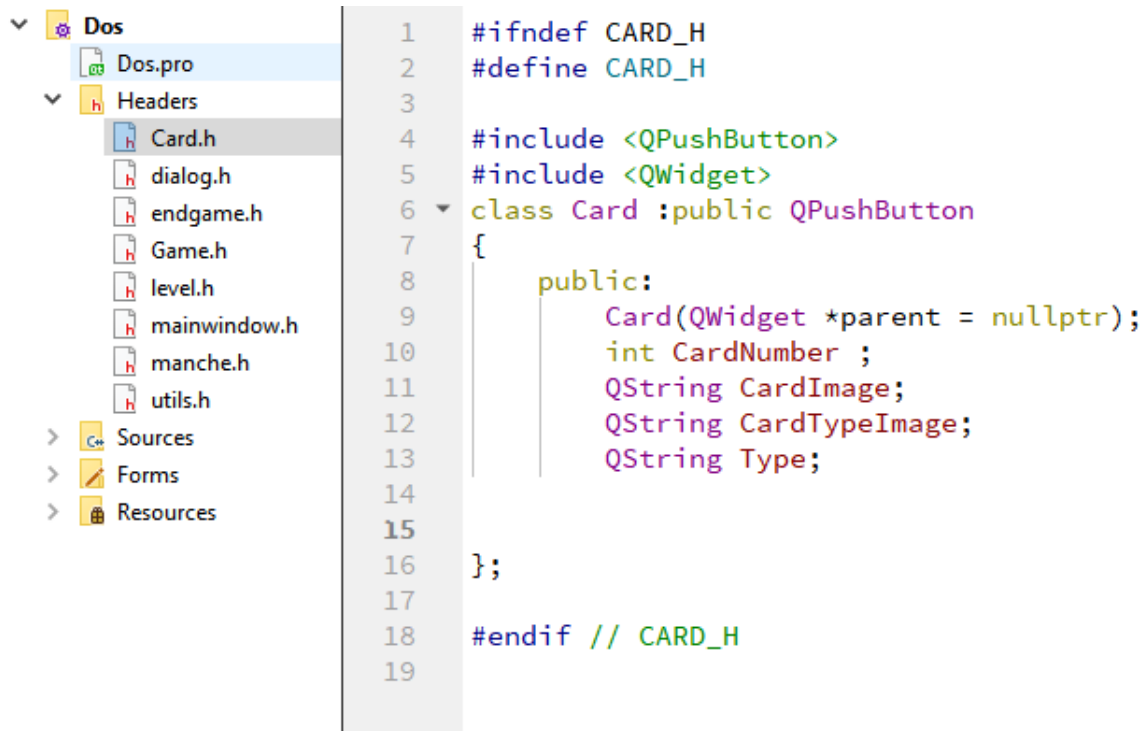
La classe dispose d'un destructeur pour libérer les ressources à la fin de l'utilisation de l'objet.

Attributs :

La classe contient plusieurs attributs pour stocker des informations sur la disposition des cartes, les cartes des joueurs, les cadres d'images, des variables de jeu, etc.

En résumé, la classe Game semble gérer de manière complète la logique de jeu, l'affichage graphique, et les interactions entre l'utilisateur et l'IA. Elle utilise les fonctionnalités de QGraphics et Qt pour créer une interface utilisateur interactive

- **Classe carte**



- ✓ **Analyse de cette classe :**

Inclusion des bibliothèques et directives de préprocesseur

les directives de préprocesseur (#ifndef, #define, #endif) sont utilisées pour éviter les inclusions multiples.

Déclaration de la classe Card :

La classe Card est déclarée en héritant de la classe QPushButton.

Membres publics de la classe Card :

La classe contient trois membres publics :

int CardNumber : Variable entière pour stocker le numéro de la carte.

QString CardImage : Chaîne de caractères pour stocker le chemin de l'image de la carte.

QString CardTypeImage : Chaîne de caractères pour stocker le chemin de l'image représentant le type de la carte.

QString Type : Chaîne de caractères pour stocker le type de la carte.

Constructeur de la classe Card :

Le constructeur de la classe Card prend un pointeur vers un objet QWidget en paramètre, avec une valeur par défaut de nullptr.

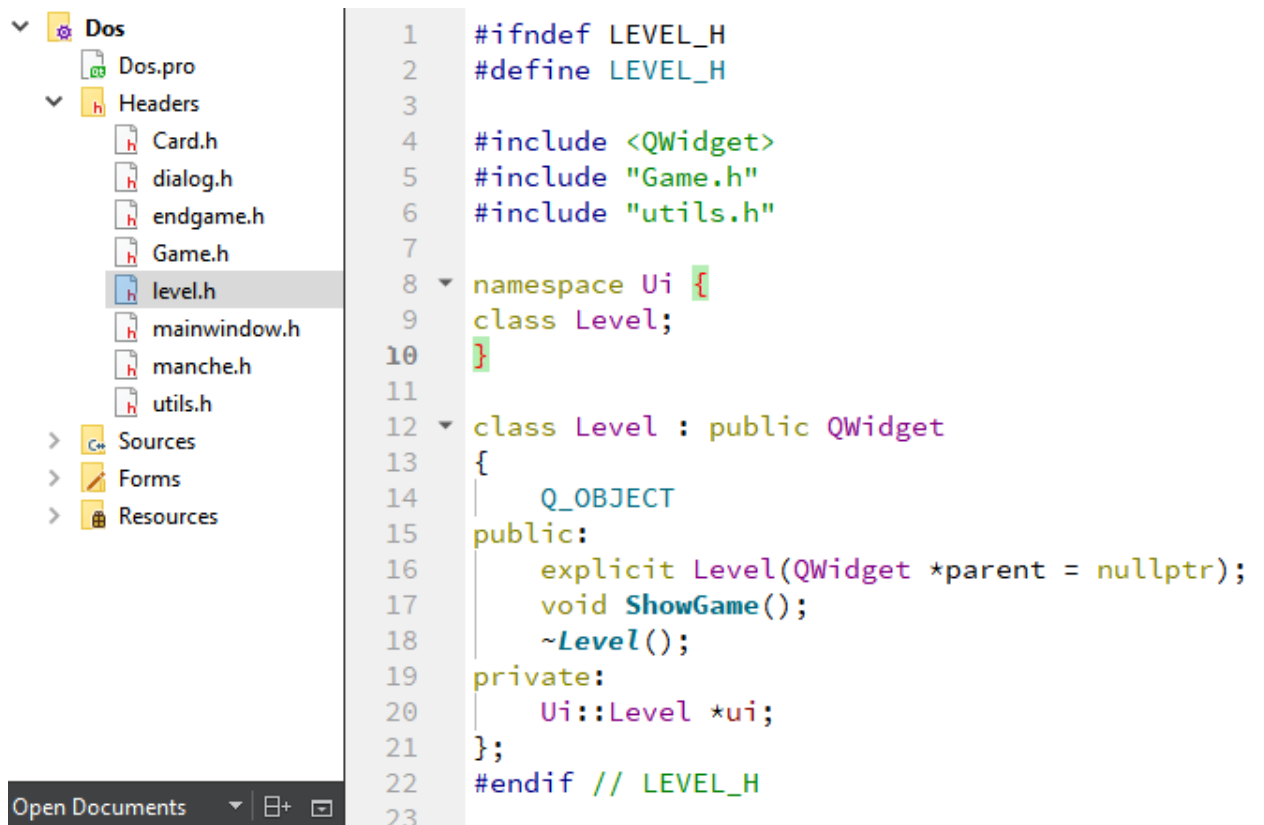
Cela signifie qu'il peut être utilisé avec ou sans parent.

Le constructeur initialise la carte en appelant le constructeur de la classe de base QPushButton et effectue d'autres initialisations propres à la classe Card.

Fin de la directive de préprocesseur :

La directive #endif marque la fin de la directive de préprocesseur #ifndef.

- **Classe level (niveau) :**



- ✓ **Analyse de classe :**

Bibliothèques incluses :

Inclusion des bibliothèques nécessaires, telles que QWidget (pour les interfaces graphiques), Game.h (la classe Game), et utils.h.

Déclaration de la classe Level :

La classe Level hérite de la classe QWidget du framework Qt et utilise le système de signaux et de slots (indiqué par Q_OBJECT).

Membres publics :

Constructeur explicite pour initialiser l'interface graphique du niveau.

Méthode publique ShowGame pour afficher le jeu (Game).

Membres privés :

La classe contient un pointeur vers un objet de type `Ui::Level`, qui semble être une classe générée automatiquement par l'outil de conception graphique de Qt.

Méthodes privées :

Aucune méthode privée n'est spécifiée dans la déclaration. Cependant, cela pourrait être défini dans la définition de la classe (le fichier `.cpp` associé).

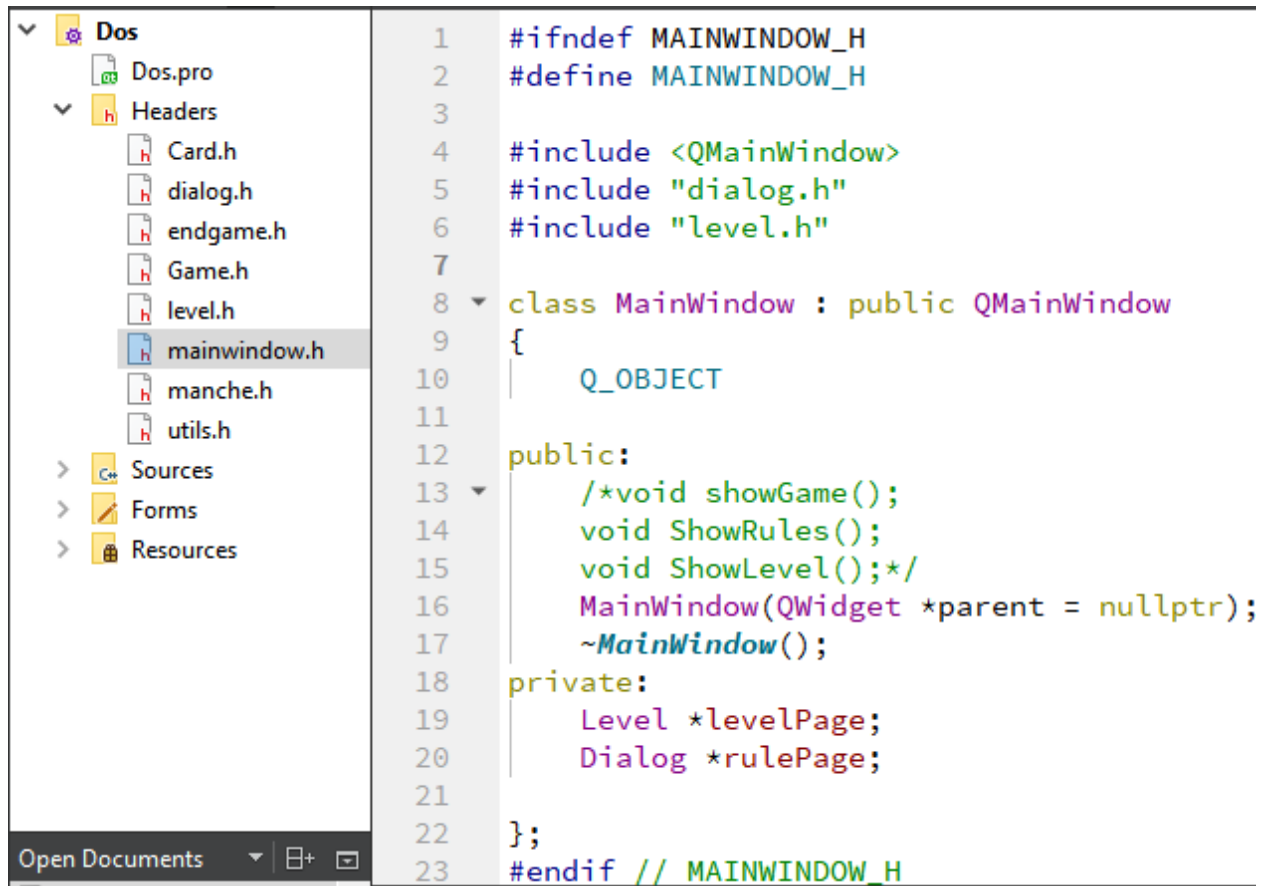
Destructeur :

La classe dispose d'un destructeur pour libérer les ressources à la fin de l'utilisation de l'objet.

Attributs :

Un pointeur vers un objet `Ui::Level`, qui est probablement utilisé pour la gestion de l'interface utilisateur via l'outil de conception graphique de Qt.

- **Classe MainWindow :**



Déclaration de la classe MainWindow :

La classe MainWindow hérite de la classe QMainWindow du framework Qt et utilise le système de signaux et de slots (indiqué par Q_OBJECT).

Membres publics :

Constructeur qui prend un pointeur vers un objet QWidget (parent) en paramètre (le parent par défaut est nullptr).

Méthode publique showGame (commentée) qui semble être destinée à afficher le jeu.

Méthode publique ShowRules (commentée) qui pourrait être destinée à afficher les règles du jeu.

Méthode publique ShowLevel (commentée) qui pourrait être destinée à afficher la sélection du niveau.

Membres privés :

La classe contient deux pointeurs vers des objets de type Level et Dialog. Ces pointeurs semblent être des pages ou des éléments d'interface graphique que la fenêtre principale peut afficher.

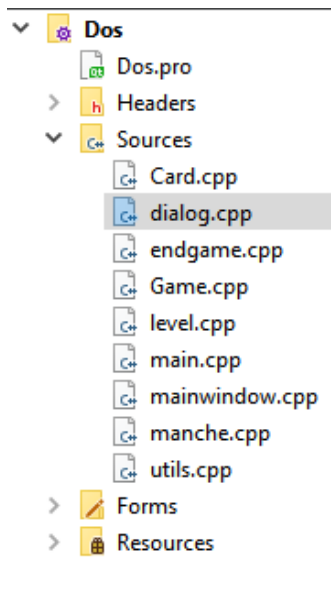
Destructeur :

La classe dispose d'un destructeur pour libérer les ressources à la fin de l'utilisation de l'objet.

En résumé, la classe MainWindow semble être responsable de la gestion de la fenêtre principale de l'application. Elle contient des méthodes pour afficher différentes pages (potentiellement les règles du jeu, le niveau, etc.) et utilise les classes Level et Dialog pour représenter ces pages spécifiques dans l'interface utilisateur.

Les fonctions/méthodes :

- Fichiers du code source du jeu :

	<pre>1 #include "dialog.h" 2 #include "ui_dialog.h" 3 4 Dialog::Dialog(QWidget *parent) : 5 QDialog(parent), 6 ui(new Ui::Dialog) 7 { 8 // fenetre regle du jeu cree avec qt design 9 this->setFixedHeight(700); 10 ui->setupUi(this); 11 } 12 13 14 Dialog::~Dialog() 15 { 16 delete ui; 17 }</pre>
--	--

❖ Fichier Principale : Game.cpp

✓ Analyse du code :

Le code commence par l'inclusion des fichiers d'en-tête (headers) et des bibliothèques du Framework Qt

Initialisation de la fenêtre principale (QMainWindow) :

La classe Game hérite de la classe QMainWindow. Le constructeur initialise la fenêtre principale avec le parent passé en paramètre (QWidget *parent).

Définition du style de fond :

La fenêtre principale est configurée avec un style de fond défini par `this->setStyleSheet("background-color :#268335");`.

Création des éléments graphiques principaux :

Un nouvel objet de type QWidget appelé Window est créé comme conteneur principal de la fenêtre.

Deux labels (nbrCartePlayer et nbrCartela) sont créés pour afficher le nombre de cartes du joueur et de l'IA.

Obtention de la taille de l'écran :

La taille de l'écran est obtenue à l'aide de `QGuiApplication::primaryScreen()` et stockée dans les variables `ScreenHeight` et `ScreenWidth`.

Chargement de l'image de fond pour les cartes :

Le chemin d'accès à l'image de fond des cartes est défini en utilisant `QCoreApplication::applicationDirPath()` et stocké dans la variable `ImageBack`.

Création des éléments graphiques pour l'affichage du score :

Un cadre ScoreBox est créé pour afficher les scores. Les labels ScoreBanner sont créés pour afficher le score de l'IA et du joueur.

Configuration des propriétés graphiques des éléments :

Les propriétés graphiques, comme la taille, la couleur, et la position, sont définies pour les différents éléments, notamment CardStatic, CardTypeImage, CardMiddleImage, ScoreBanner, nbrCartela, et nbrCartePlayer.

Positionnement des éléments graphiques dans la fenêtre

Les éléments graphiques sont positionnés dans la fenêtre principale en utilisant différentes méthodes de positionnement, telles que move, setAlignment, et setFixedSize.

Appel de méthodes pour l'initialisation des cartes :

La méthode InitCard() est appelée pour initialiser la liste des cartes.

Les méthodes FirstAnimation(true) et FirstAnimation(false) sont appelées pour distribuer les 5 cartes à l'IA et au joueur.

Définition du widget central :

Le widget Window est défini comme widget central de la fenêtre principale en utilisant setCentralWidget(Window).

En résumé, ce constructeur initialise la fenêtre principale du jeu en créant et positionnant divers éléments graphiques, en définissant le style visuel, et en appelant des méthodes pour l'initialisation des cartes. Il configure également l'affichage des scores et des éléments visuels tels que les cartes.

Fonction SetPosition sert à définir les positions des cartes, il y a 11 place de carte disponible pour l'IA et aussi 11 place de carte possible pour l'utilisateur.

Fonction FirstAnimation est effectué à la toute première animation lors du lancement du jeu. Elle distribue 5 cartes à l'IA et 5 cartes à l'utilisateur.

Fonction UserPlayer sert à connecter les clics de l'utilisateur sur les différentes cartes qu'il a.

Fonction Player contrôle les actions d'un utilisateur, notamment s'il doit prendre une carte, et aussi pour connecter les nouvelles cartes jouées par l'utilisateur à sa liste de cartes.

Fonction Animation afin d'avoir une animation pour les cartes jouées par l'utilisateur ou par l'adversaire (AI)

Fonction CanUserPlayIndex : pour calculer la liste des index jouable par l'utilisateur. Elle renvoie une liste d'index des cartes qui peuvent être jouées

Fonction InitCard pour créer et initialiser la liste des 40 cartes

- **Fichier mainwindow.cpp (l'interface)**

- ✓ **Analyse du code :**

Constructeur MainWindow :

La classe MainWindow hérite de la classe QMainWindow de Qt.

Le constructeur initialise la fenêtre principale avec un parent (l'objet QWidget parent).

La fenêtre principale a un style de fond défini par this->setStyleSheet("background-color :#268335");.

Création des éléments graphiques :

Un nouvel objet de type QWidget appelé MyWindow est créé comme conteneur principal de la fenêtre.

Les pages rulePage (de type Dialog) et levelPage (de type Level) sont instanciées.

Des éléments graphiques tels que Branding, BrandingCard, des boutons (Rules et level), et un label (NameDos) sont créés pour composer l'interface.

Positionnement des éléments dans un layout :

Un QVBoxLayout est utilisé pour organiser les éléments verticalement dans MyWindow.

Un QHBoxLayout appelé ButtunBox est utilisé pour organiser horizontalement les boutons "Rules" et "level".

Les éléments sont ajoutés au layout avec des ajustements de taille et d'alignement.

Attribution des images de fond :

Les images de fond pour Branding et BrandingCard sont définies à partir de fichiers locaux avec setStyleSheet et l'utilisation de chemins d'accès relatifs.

Définition des styles des boutons :

Les boutons "Rules" et "level" ont des styles définis avec des feuilles de style CSS. Cela inclut des changements de couleur lorsqu'ils sont survolés.

Connexions des signaux et des slots :

Deux connexions sont établies avec les boutons "Rules" et "level" à l'aide de la fonction connect. Lorsque l'utilisateur clique sur le bouton "level", la page du niveau (levelPage) est affichée et la fenêtre principale est fermée (this->close()). Lorsque l'utilisateur clique sur le bouton "Rules", la page des règles (rulePage) est affichée.

Affichage de la fenêtre principale :

Enfin, MyWindow est défini comme widget central de la fenêtre principale à l'aide de setCentralWidget.

Destructeur ~MainWindow :

Le destructeur libère la mémoire allouée pour les pages levelPage et rulePage avec l'opérateur delete.

En résumé, la classe MainWindow est responsable de la création de l'interface utilisateur principale de l'application, avec des boutons pour accéder aux règles du jeu et à la sélection du niveau. Elle gère également les transitions entre les pages en établissant des connexions entre les signaux et les slots.

- Fichier manche.cpp :

```
1  #include "manche.h"
2  #include "ui_manche.h"
3
4  Manche::Manche(QWidget *parent) :
5      QWidget(parent),
6      ui(new Ui::Manche)
7  {
8      // fenetre de manche du jeu cree avec qt design
9      this->setFixedSize(500,600);
10     ui->setupUi(this);
11
12     if(IaScore < PlayerScore){
13         QMovie *gif = new QMovie(":/assets/93dx.gif");
14         ui->label->setFixedSize(212,256);
15         ui->label->setMovie(gif);
16         ui->label->setScaledContents(true);
17         ui->label_2->setText("Alors, c'est qui le boss ?\nVous avez gagné la manche");
18         ui->label_2->setAlignment(Qt::AlignCenter);
19         gif->start();
20     }else{
21         QMovie *gif = new QMovie(":/assets/93c5.gif");
22         ui->label->setFixedSize(212,256);
23         ui->label->setMovie(gif);
24         ui->label->setScaledContents(true);
25         ui->label_2->setText("Oupss Vous avez perdu la manche.\nMais la partie continue");
26         ui->label_2->setAlignment(Qt::AlignCenter);
27
28         gif->start();
29     }
30
31     connect(ui->pushButton,&QPushButton::clicked,this,[&]() {
32         if(nbrManche >= 1){
33             ShowPage();
34         };
35         nbrManche--;
36     });
37
38     void Manche::ShowPage(){
39         Game *secondGamePage = new Game;
40         secondGamePage->showMaximized();
41         this->close();
42         this->deleteLater();
43     }
44
45     Manche::~~Manche()
46     {
47         delete ui;
48     }
49
```

✓ Analyse du code :

Constructeur Manche :

La classe Manche hérite de la classe QWidget de Qt. Le constructeur initialise la fenêtre de la manche avec une taille fixe de 500x600 pixels en utilisant `this->setFixedSize(500,600)`.

Le constructeur utilise un fichier d'interface utilisateur généré par l'outil de conception graphique de Qt (`ui->setupUi(this);`).

Affichage conditionnel de l'interface :

En fonction des scores (`laScore` et `PlayerScore`), une animation GIF différente est affichée, accompagnée d'un texte descriptif dans `ui->label_2`.

L'animation GIF est chargée à partir de ressources externes ("`:/assets/93dx.gif`" et "`:/assets/93c5.gif`") et est affichée dans `ui->label`. La taille de `ui->label` est fixée à 212x256 pixels.

Connecteur pour le bouton "Continuer" :

Un connecteur est établi entre le bouton `ui->pushButton` et une fonction lambda. Lorsque le bouton est cliqué, la fonction lambda est exécutée. Cette fonction vérifie si le nombre de manches restantes (`nbrManche`) est supérieur ou égal à 1. Si oui, elle appelle la fonction `ShowPage()`.

Fonction ShowPage :

La fonction ShowPage crée une nouvelle instance de la classe Game appelée secondGamePage et l'affiche maximisée. Ensuite, elle ferme la fenêtre de la manche actuelle (`this->close()`) et supprime l'instance actuelle (`this->deleteLater()`).

Destructeur ~Manche :

Le destructeur libère la mémoire allouée pour l'interface utilisateur (UI) avec l'opérateur delete.

En résumé, la classe Manche représente la fenêtre de fin de manche dans le jeu. Elle affiche une animation GIF et un message en fonction des scores, offre un bouton pour passer à la manche suivante, et gère la transition vers la page du jeu (Game) lorsque le joueur décide de continuer.

**Merci infiniment Madame pour votre
Attention**