24/12/2023

RAPPORT
DU JEU

PROJET : JEU DE CARTES MAROCAIN EN C++ ET QT

Encadré par : Mme. BEN ABDELOUAHAB Ikrame

et Mr. EL AACHAK Lotfi

Réalisé par les étudiantes : BAKALI KORAMI Aya

AABOUD Nissrin

FSTT 2023-2024

Le présent rapport documente le développement d'un jeu de cartes sous le nom "Kbir Yakol Sghir". Ce projet a été réalisé dans le cadre du cours de programmation en utilisant le langage C++ et le framework Qt.

L’objectif principal c’est de mettre en pratique les concepts de la programmation orientée objet (poo) pour créer une application

Introduction du jeu :

"Kbir Yakol Sghir" est un jeu de cartes populaire au Maroc. Il se joue avec un jeu de 40 cartes, avec des valeurs allant de 1 à 10. L'objectif du jeu est de remporter toutes les cartes afin de gagner le jeu.

Chaque joueur débute avec 5 cartes, cherchant à accumuler des points en remportant des manches. Les cartes sont jouées alternativement sur le tapis virtuel, la carte avec la valeur numérique la plus élevée gagnant la manche. La carte spéciale "carte numéro 1 triomphe de toutes les autres. En cas de confrontation avec le même numéro, les deux cartes initialement posées restent sur le tapis, et chaque joueur ajoute deux nouvelles cartes de leur main. Le gagnant est déterminé par le joueur dont la somme des numéros des nouvelles cartes est la plus élevée.

Règles du Jeu :

1. **Distribution des Cartes :** Chaque joueur reçoit 5 cartes d'un jeu.
2. **Début de la Manche :** Les joueurs posent tour à tour une carte sur le tapis virtuel.
3. **Comparaison des Cartes :** La carte avec la valeur numérique la plus élevée remporte la manche, sauf si la carte posé est "la carte dont le numéro 1" , elle remporte automatiquement n'importe quelle autre carte .
4. **Situation Spéciale - Même Numéro :** En cas de confrontation avec le même numéro, les deux cartes initialement posées restent sur le tapis. Chaque joueur ajoute deux nouvelles cartes de leur main à côté des cartes initialement posées, face cachée.
5. **Révélation des Nouvelles Cartes :** Les nouvelles cartes sont retournées, et le gagnant est celui dont la somme des numéros

des nouvelles cartes est la plus élevée, en prend en considération la condition de la « carte dont le numéro est 1 », le gagnant dans ce cas remporte tous les cartes posé dans le tapis .

6. **Gagnant de la Manche :** Le joueur remportant la manche en gagnant les deux cartes jouées marque un point.
7. **Calcul des Points :** À la fin de chaque manche, les points sont attribués. Chaque fois qu'un joueur gagne les deux cartes de la manche, il marque un point.
8. **Spécialité de la Carte 1 :** La carte dont le numéro est 1, remporte automatiquement n'importe quelle autre carte, indépendamment de sa valeur numérique.
9. **Fin de la Partie :** Le jeu continue jusqu'à épuisement des cartes. Le joueur avec le plus grand nombre de points est déclaré comme gagnant du jeu .

Architecture du Projet :

Pour la première interface :

- Nous avons utilisée **QStackedWidget** car nous avons plusieurs pages dans notre application et il nous permet de passer facilement d'une vue à une autre.

Pour l'intégration des images dans l'interface, nous avons utiliser le fichier de ressources (par exemple, "image.qrc") pour référencer l'image dans notre projet QT.
N'oublions pas l'utilisation du **QLabel** afin d'afficher du texte et des images. Il simplifie le processus d'affichage d'une image en offrant des méthodes et des propriétés dédiées pour gérer les images de manière efficace , par exemple il s'adapte automatiquement à la taille de l'image qu'il contient. Cela

permet une gestion plus facile de l'affichage, en évitant des ajustements manuels de la taille du widget.


```
22
23 # Default rules for deployment.
24 qnx: target.path = /tmp/$${TARGET}/bin
25 else: unix:!android: target.path = /opt/$${TARGET}/bin
26 !isEmpty(target.path): INSTALLS += target
27
28 RESOURCES += \
29     BG.qrc \
30     image.qrc
```

- Pour les boutons :
  Pour passer d'une page à une autre on utilise la fonction suivante :

```
void MainWindow::on_pushButton_clicked()
{
    ui->stackedWidget->setCurrentIndex(1);
}
```

void MainWindow::on_pushButton_clicked(): C'est la déclaration du slot associé au clic sur le bouton. Il est automatiquement appelé lorsque le bouton associé (**pushButton**) est cliqué.
ui->stackedWidget: **ui** est un pointeur vers l'interface utilisateur, et **stackedWidget** est le nom du QStackedWidget dans votre interface.
setCurrentIndex(1): C'est une méthode de QStackedWidget. Elle prend un index en paramètre et change la page affichée à celle correspondant à cet index.
Alors lorsque on clique sur le bouton associé, la page actuellement affichée dans le QStackedWidget passe à la deuxième page (index 1).

pour qu'on peut sortir du jeu :

```
void MainWindow::on_pushButton_2_clicked()
{
    QApplication::quit();
}
```

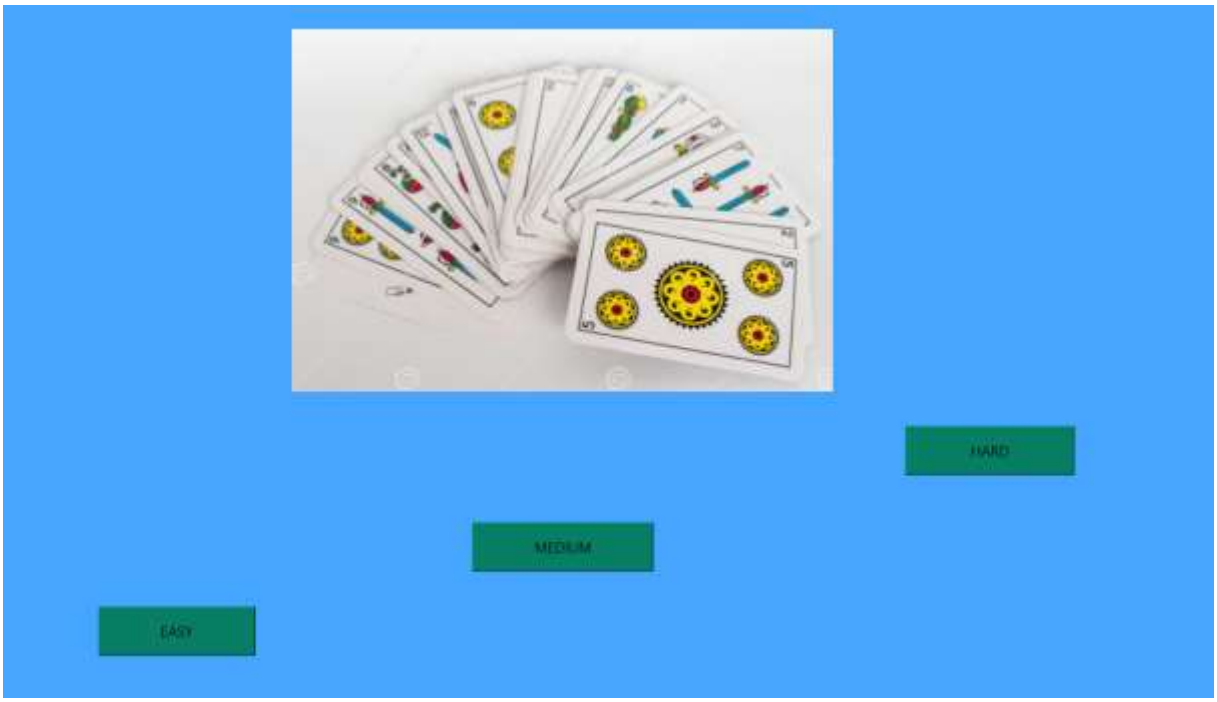
void MainWindow::on_pushButton_2_clicked(): C'est la signature d'une fonction membre de la classe **MainWindow** dans laquelle nous avons connecté l'événement de clic du bouton **pushButton_2**. Lorsque ce bouton est cliqué, cette fonction sera exécutée.

L'appel à **QApplication::quit()** entraîne la fermeture de l'application en cours d'exécution. Cela signifie que l'application se termine et que le programme quitte.

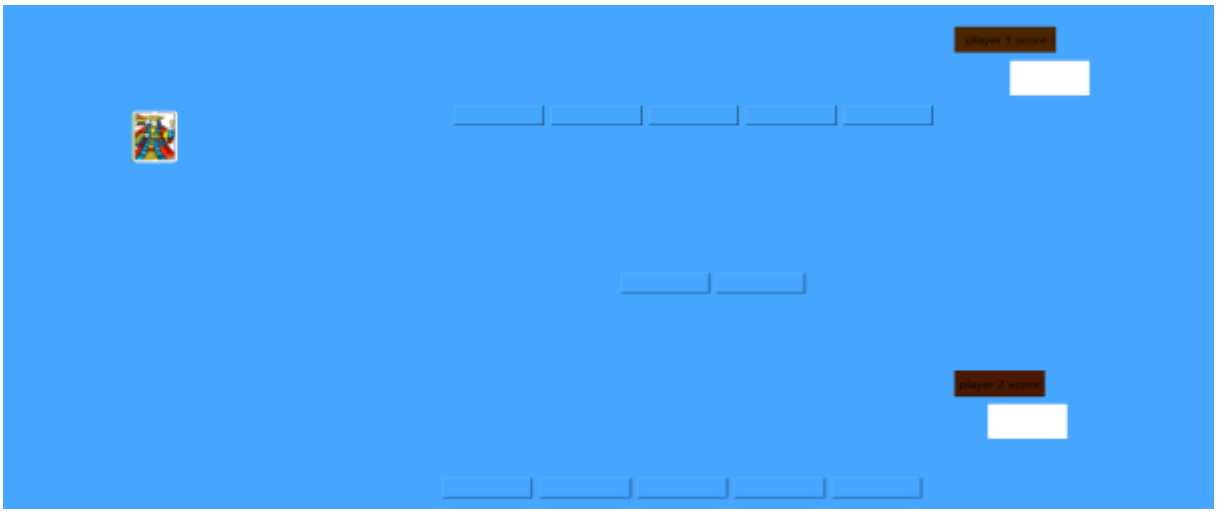
En bref lorsque on clique sur le bouton associé (**pushButton_2**), la fonction **on_pushButton_2_clicked()** est appelée, et elle, à son tour, appelle **QApplication::quit()**, ce qui provoque la fermeture de l'application.

- Pour la deuxième page :
  Dans notre implémentation du jeu "Kbir Yakol Sghir", nous avons pris soin de spécifier trois niveaux distincts, chacun offrant une expérience de jeu unique. Ces niveaux sont définis en fonction de la difficulté globale du jeu, avec des distinctions Facile, Moyen et Difficile. Chaque niveau présente des variations spécifiques dans les règles, la complexité des cartes et la compétitivité de l'intelligence artificielle. Cette diversité permet aux joueurs de personnaliser leur expérience en choisissant le niveau qui correspond à leurs compétences et à leurs préférences. Que ce soit pour les débutants souhaitant apprendre les bases du jeu, les joueurs intermédiaires cherchant un équilibre entre stratégie et divertissement, ou les experts

recherchant un défi intense, notre jeu propose une aventure adaptée à tous les niveaux de compétence.



- Pour la troisième page :



On a d’abord stocké les cartes en utilisant la fonction pixmap :

```
// Chargez les images et associez-les au bouton pushButton_17

cardImages.push_back(QPixmap(":/image/00.gif"));
cardImages.push_back(QPixmap(":/image/01.gif"));
cardImages.push_back(QPixmap(":/image/02.gif"));
cardImages.push_back(QPixmap(":/image/03.gif"));
cardImages.push_back(QPixmap(":/image/04.gif"));
cardImages.push_back(QPixmap(":/image/05.gif"));
cardImages.push_back(QPixmap(":/image/06.gif"));
cardImages.push_back(QPixmap(":/image/07.gif"));
cardImages.push_back(QPixmap(":/image/08.gif"));
cardImages.push_back(QPixmap(":/image/09.gif"));
cardImages.push_back(QPixmap(":/image/10.gif"));
cardImages.push_back(QPixmap(":/image/11.gif"));
cardImages.push_back(QPixmap(":/image/12.gif"));
cardImages.push_back(QPixmap(":/image/13.gif"));
cardImages.push_back(QPixmap(":/image/14.gif"));
```

La fonction **QPixmap** dans Qt est utilisée pour charger, afficher manipuler des images et des graphiques.

Aussi la fonction push_back est utilisé pour étendre dynamiquement le vecteur cardImages en ajoutant de nouvelles images à mesure qu'elles sont chargées. Cela est particulièrement utile lorsque le nombre d'images n'est pas connu à l'avance, et le vecteur doit s'ajuster automatiquement pour accommoder les nouvelles données.

- La fonction pushbutton :

```cpp
void MainWindow::on_pushButton_17_clicked()
{
    qDebug() << "PushButton_17 clicked";

    // Vecteur de boutons à mettre à jour
    QVector<QPushButton *> targetButtons = {ui->pushButton_14, ui->pushButton_7, ui->pushButton_
                                            ui->pushButton_13, ui->pushButton_15, ui->pushButton_

    // Distribuez les images aux boutons
    for (int i = 0; i < targetButtons.size(); ++i)
    {
        for (int j = 0; j < 4; ++j)
        {
            int index = i * 4 + j; // Calcul de l'index dans le vecteur cardImages
            if (index < cardImages.size())
            {
                targetButtons[i]->setIcon(QIcon(cardImages[index]));
                targetButtons[i]->setIconSize(targetButtons[i]->size());
            }
        }

        // Animation des boutons
        QPropertyAnimation *animation = new QPropertyAnimation(targetButtons[i], "pos");
        animation->setDuration(500); // Durée de l'animation en millisecondes
        animation->setStartValue(QPoint(targetButtons[i]->x(), targetButtons[i]->y() - 700)); //
        animation->setEndValue(QPoint(targetButtons[i]->x(), targetButtons[i]->y()));

        // Connectez le signal finished à la suppression de l'animation
        connect(animation, &QPropertyAnimation::finished, animation, &QPropertyAnimation::deleteL

        // Démarrez l'animation
        animation->start(QPropertyAnimation::DeleteWhenStopped);
    }
}
```

Cette fonction est appelée en réponse au clic sur le bouton pushButton_17. Elle distribue également les images du vecteur cardImages, mais elle ajoute également une animation pour afficher visuellement le chargement des images.

- La fonction distributeCardImages :

```cpp
void MainWindow::distributeCardImages(QVector<QPushButton *> &buttons)
{
    qDebug() << "Distributing card images to buttons.";

    // Distribuez les images aux boutons
    for (int i = 0; i < buttons.size(); ++i)
    {
        if (i < cardImages.size())
        {
            buttons[i]->setIcon(QIcon(cardImages[i]));
            buttons[i]->setIconSize(buttons[i]->size());
        }
    }
}
```

Cette fonction a pour but de distribuer les images du vecteur cardImages sur les boutons spécifiés dans le vecteur buttons.
En résumé, la première fonction est utilisée pour une distribution simple d'images sur des boutons, tandis que la seconde fonction est utilisée pour la même distribution avec l'ajout d'une animation visuelle pour améliorer l'aspect esthétique de l'interface utilisateur.