PROJET : Jeu de Cartes Marocain en C++ et Qt

RONDA

Encadrée par : prof Ikram Ben Abdel Ouahab

Notre équipe

YASSINE JARRADI

LST IDAI

ASSAAD EL AOUZI

LST IDAI

I - Introduction :

La ronda est un jeu de cartes qui se joue avec des cartes espagnoles ou cartes italiennes (quatre couleurs latines: coupes, épées, bâtons, deniers) en utilisant quarante cartes. On trouve des cartes numérotées de 1 à 7, et trois autres cartes : le Valet (10), le Cavalier (11) et le Roi (12)). On remarquera qu'il « manque » les 8 et 9.

Le jeu se joue à deux, trois ou quatre joueurs en constituant dans le cas de 4 joueurs 2 équipes qui jouent en commun et dont les joueurs sont assis l'un en face de l'autre avec des jetons.

I - 1. L’objectife du projet

L'objectif est de ramasser le plus de cartes possible pour marquer le plus de points. La manche est gagnée par le premier joueur ou équipe qui atteint plus de 20 cartes. Quant à la partie, elle est gagnée lorsqu'une équipe totalise la somme de 41 points.

I - 2. Les règles du deu

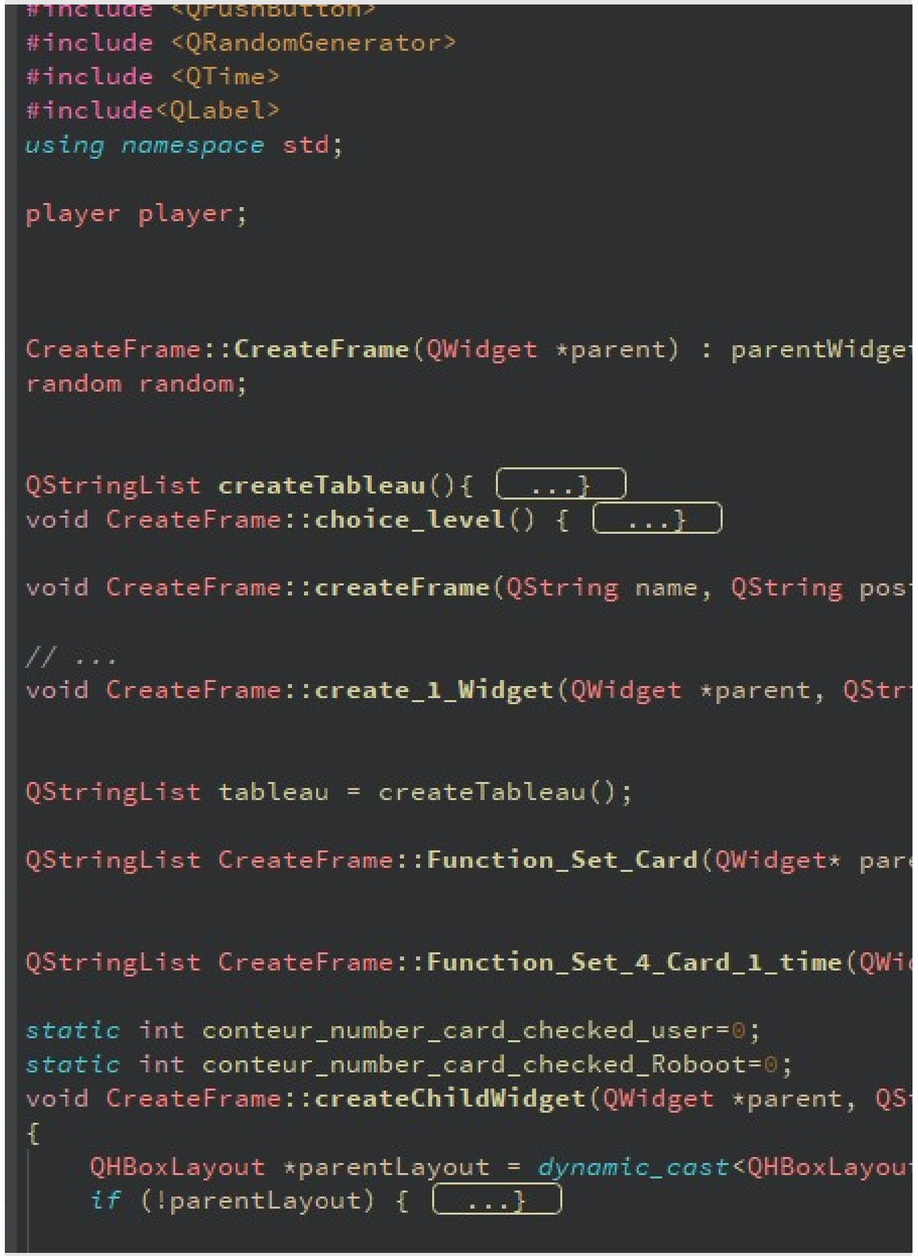
On détermine par une manière quelconque le donneur. Celui-ci distribue dans le sens inverse de l'aiguille d'une montre(de droite à gauche) quatre cartes par personne et en retournant quatre cartes sur le tapis.

On joue tour à tour,. Lorsque toutes les cartes ont été jouées, le donneur distribue à nouveau quatre cartes par personne (toujours sans mettre de cartes au milieu de la table), ceci jusqu'à épuisement du talon.

II – Choix de Conception

II -1. Architecture Globale du Code

Le jeu de cartes marocain a été implémenté en utilisant une approche orientée objet. L'architecture globale du code est structurée autour de trois classes principales : CreateFrame, Player, et Random.



```
#include <QPushButton>
#include <QRandomGenerator>
#include <QTime>
#include<QLabel>
using namespace std;

player player;



CreateFrame::CreateFrame(QWidget *parent) : parentWidget
random random;


QStringList createTableau(){ ...}
void CreateFrame::choice_level() { ...}

void CreateFrame::createFrame(QString name, QString pos

// ...
void CreateFrame::create_1_Widget(QWidget *parent, QStr


QStringList tableau = createTableau();

QStringList CreateFrame::Function_Set_Card(QWidget* par


QStringList CreateFrame::Function_Set_4_Card_1_time(QWi

static int conteur_number_card_checked_user=0;
static int conteur_number_card_checked_Roboot=0;
void CreateFrame::createChildWidget(QWidget *parent, QS
{
    QHBoxLayout *parentLayout = dynamic_cast<QHBoxLayou
    if (!parentLayout) { ...}
```

II -2. Classe CreateFrame

La classe CreateFrame est responsable de la création de l'interface utilisateur du jeu. Elle utilise la bibliothèque Qt pour la gestion des widgets et la mise en page. Le choix d'utiliser Qt a été motivé par sa facilité d'utilisation et sa robustesse pour le développement d'interfaces graphiques.

La classe CreateFrame suit le modèle MVC (Modèle-Vue-Contrôleur), où elle agit principalement en tant que contrôleur. Elle interagit avec le modèle (la classe Player) pour obtenir les informations nécessaires à l'affichage et gère les événements utilisateur provenant de l'interface.

```
nclude "player.h"
nclude "createframe.h"
nclude <QWidget>
nclude <QPushButton>
nclude <QHBoxLayout>
nclude <iostream>
nclude <cstring>
ayer::player() {}


t player::play_Set_Card( QWidget *frameCenterWidget,QPushButto
  int number_card_checked=0;

  if (frameCenterWidget) {
      QList<QWidget*> childWidgets = frameCenterWidget->findCh
      QString buttonName = button ->objectName();
      QString number_card_gamer=buttonName.mid(1, 1);
      QWidget *parentWidget = button->parentWidget();

      // Iterate through the list and print the names
      bool add = true;

      for (QWidget *childWidget : childWidgets) {
          QString widgetName = childWidget->objectName();
          QString number_card_in_tabel=widgetName.mid(1, 1);
          if(number_card_gamer.compare(number_card_in_tabel) ==
              childWidget->deleteLater();
              parentWidget->deleteLater();
              number_card_checked=2;
              add=false;

              break;

          } ;
      }
```

II -3. Classe Player

La classe Player représente un joueur dans le jeu. Elle encapsule les données relatives à chaque joueur, telles que les cartes en main, le score, etc. L'utilisation de cette classe permet une encapsulation efficace des informations liées à chaque joueur, favorisant la modularité et la maintenabilité du code.

Le modèle MVC est également appliqué au niveau de la classe Player. La classe agit en tant que modèle, contenant les données, tandis que la logique de contrôle associée aux actions du joueur est gérée par la classe CreateFrame.

II -4. Classe Random

La classe Random est utilisée pour générer des valeurs aléatoires, notamment pour mélanger les cartes au début du jeu. Cette séparation des préoccupations permet d'isoler la génération aléatoire, favorisant une meilleure modularité du code.

II -5. Explication des Choix de Conception

Les choix de conception ont été orientés vers la simplicité, la modularité et la maintenabilité du code. L'utilisation du modèle MVC facilite la séparation des responsabilités, ce qui rend chaque classe plus indépendante et réutilisable.
Le recours à Qt pour l'interface utilisateur offre une expérience de développement graphique plus conviviale tout en garantissant une base solide pour la gestion des composants graphiques.
La classe Random a été introduite pour centraliser la génération aléatoire, suivant le principe du "Single Responsibility Principle", ce qui rend le code plus lisible et extensible.
En résumé, l'architecture choisie vise à fournir une base solide pour l'expansion future du jeu tout en rendant le code plus compréhensible et modifiable.

III - Diagramme de Classes

III - 1.Explication des Classes :

III - 1.1.CreateFrame :

- Rôle : Cette classe agit en tant que contrôleur dans le modèle MVC. Elle est responsable de la création et de la gestion de l'interface utilisateur à l'aide de la bibliothèque Qt.
- Responsabilités : Gestion des événements utilisateur, mise à jour de l'interface en réponse aux actions des joueurs, liaison avec la classe Player pour obtenir les informations nécessaires.

III - 1.2 player:

- **Rôle :** Représente un joueur dans le jeu de cartes marocain.
- **Responsabilités :** Stockage des informations spécifiques à chaque joueur telles que les cartes en main, le score, etc. Implémente la logique du joueur, y compris les actions possibles comme la pioche de cartes et le calcul du score.

III - 1.3. Random:

- Rôle : Génère des valeurs aléatoires pour les besoins du jeu, tels que le mélange des cartes.
- Responsabilités : Fournir des méthodes pour générer des nombres aléatoires utilisés dans différentes parties du code.

IV - Interaction Utilisateur

L'interaction de l'utilisateur avec le jeu se fait principalement à travers l'interface utilisateur créée par la classe CreateFrame. Voici comment les actions de l'utilisateur sont gérées dans le code :

Clic de Souris :

- Lorsqu'un joueur clique sur une carte, l'événement est capturé par une methode dans la classe CreateFrame.
- La classe CreateFrame détermine quel joueur a effectué l'action en cours, puis informe la classe Player associée du joueur correspondant.

Mise à Jour de l'Interface :

- Après chaque action de l'utilisateur, la classe CreateFrame met à jour l'interface pour refléter les changements, tels que la mise à jour des cartes en main, le score, etc.
- Les signaux émis par la classe Player sont utilisés pour déclencher ces mises à jour, suivant le modèle MVC.

En résumé, l'interaction utilisateur est gérée de manière à assurer une expérience fluide et intuitive. Les actions de l'utilisateur déclenchent des événements qui sont capturés par la classe CreateFrame, puis relayés aux instances appropriées de la classe Player pour effectuer les actions de jeu correspondantes.

V - Difficultés Rencontrées

V - 1. Problèmes Techniques:

V - 1.1.Intégration de Qt avec le Modèle de Données:

L'une des difficultés techniques auxquelles nous avons été confrontés était l'intégration harmonieuse de Qt avec le modèle de données représenté par la classe **Player**. En particulier, la mise à jour dynamique de l'interface utilisateur en réponse aux changements de l'état du joueur n'était pas immédiatement évidente.
Pour surmonter ce défi, nous avons exploré les signaux et les slots de Qt. Les signaux émis par la classe **Player** sont connectés aux slots dans la classe **CreateFrame**, permettant une mise à jour automatique de l'interface utilisateur chaque fois qu'un joueur effectue une action, telle que la pioche d'une carte.

V - 1.2.Gestion des Événements Utilisateur :

La gestion efficace des événements utilisateur, tels que les clics de souris et les saisies clavier, a été un autre point délicat. Nous voulions nous assurer que les actions de l'utilisateur étaient correctement interprétées et traitées, en particulier dans le contexte d'un jeu de cartes où la réactivité est cruciale.
La solution a été trouvée en exploitant les capacités de gestion d'événements de Qt. Les événements associés à l'interface utilisateur sont interceptés par la classe **CreateFrame**, qui les achemine vers le modèle (classe **Player**) pour une manipulation appropriée.

VI - Améliorations Possibles

VI - 1.Fonctionnalités à Ajouter

.

VI - 1.1.Mode Multijoueur en Réseau :

Une amélioration significative serait l'ajout d'un mode multijoueur en réseau, permettant à plusieurs joueurs de participer à une partie en ligne. Cela nécessiterait une extension du modèle **Player** pour gérer les interactions entre les joueurs distants.

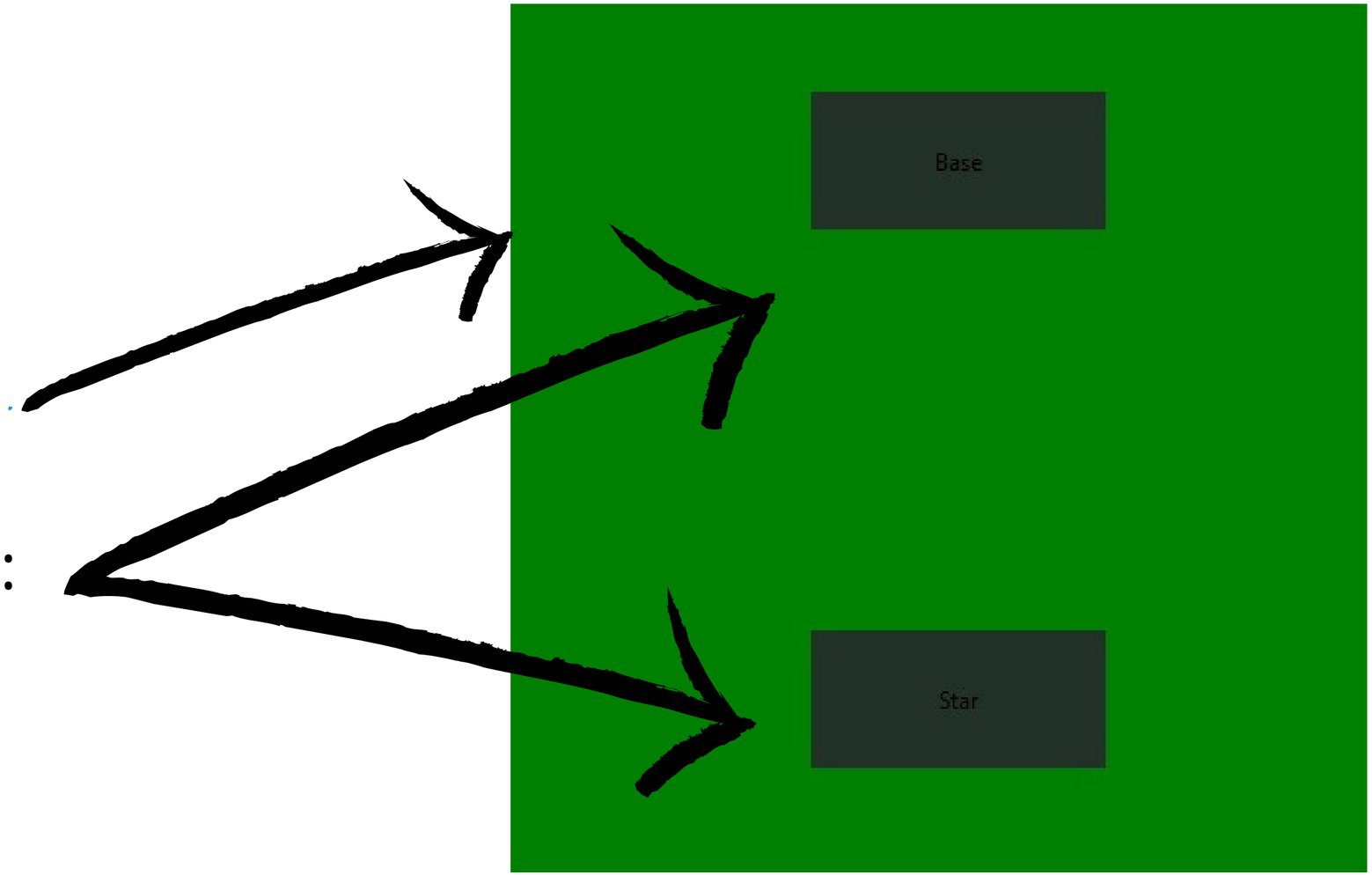
VI - 1.2.Intelligence Artificielle Améliorée :

L'intégration d'une intelligence artificielle (IA) plus avancée pour simuler des adversaires virtuels offrirait une expérience de jeu plus stimulante. Cela impliquerait des ajustements dans la logique de la classe Player et l'ajout d'une nouvelle classe pour la gestion de l'IA.

Des screenshots du jeu:

page Accueil:

Les diffucultés :

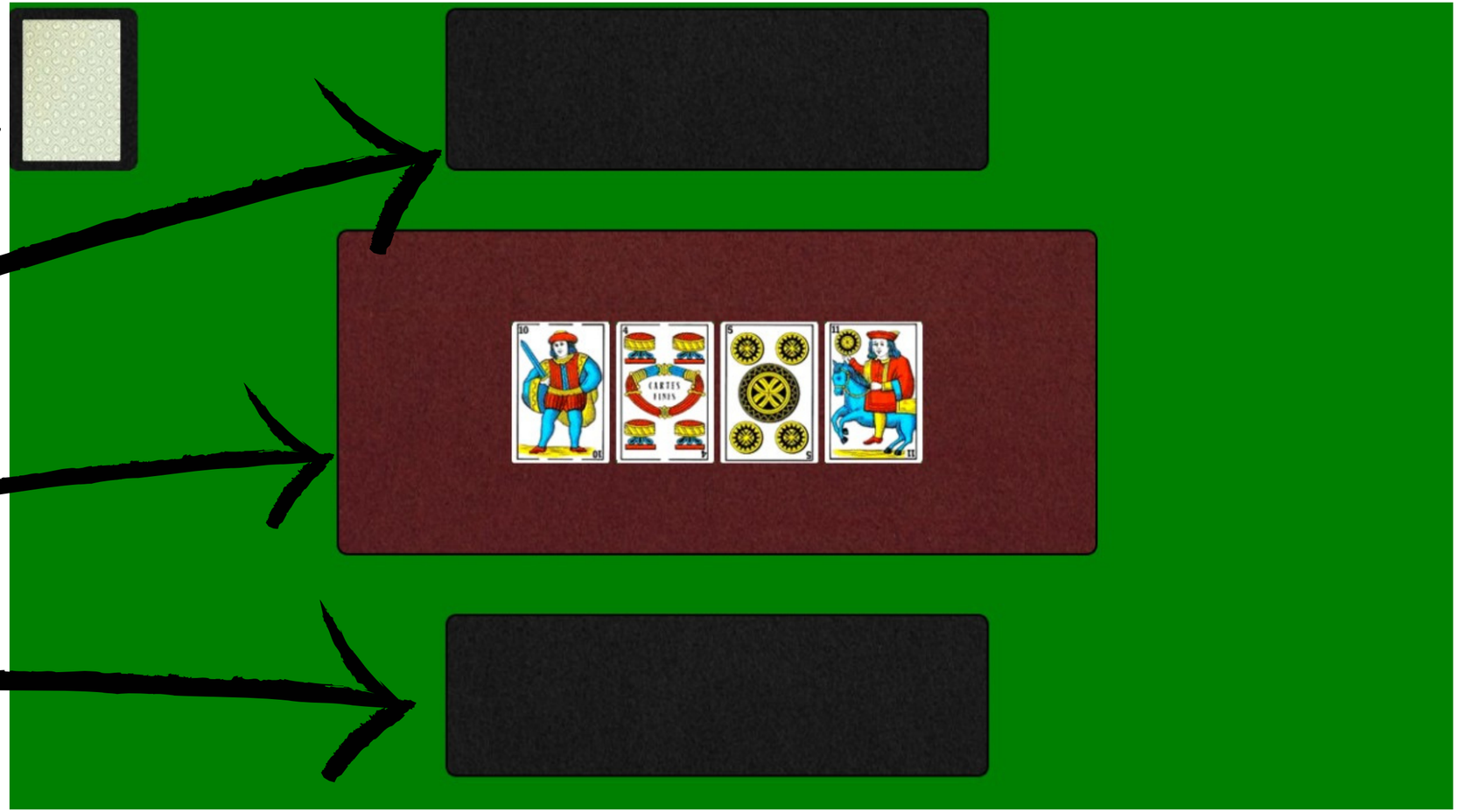


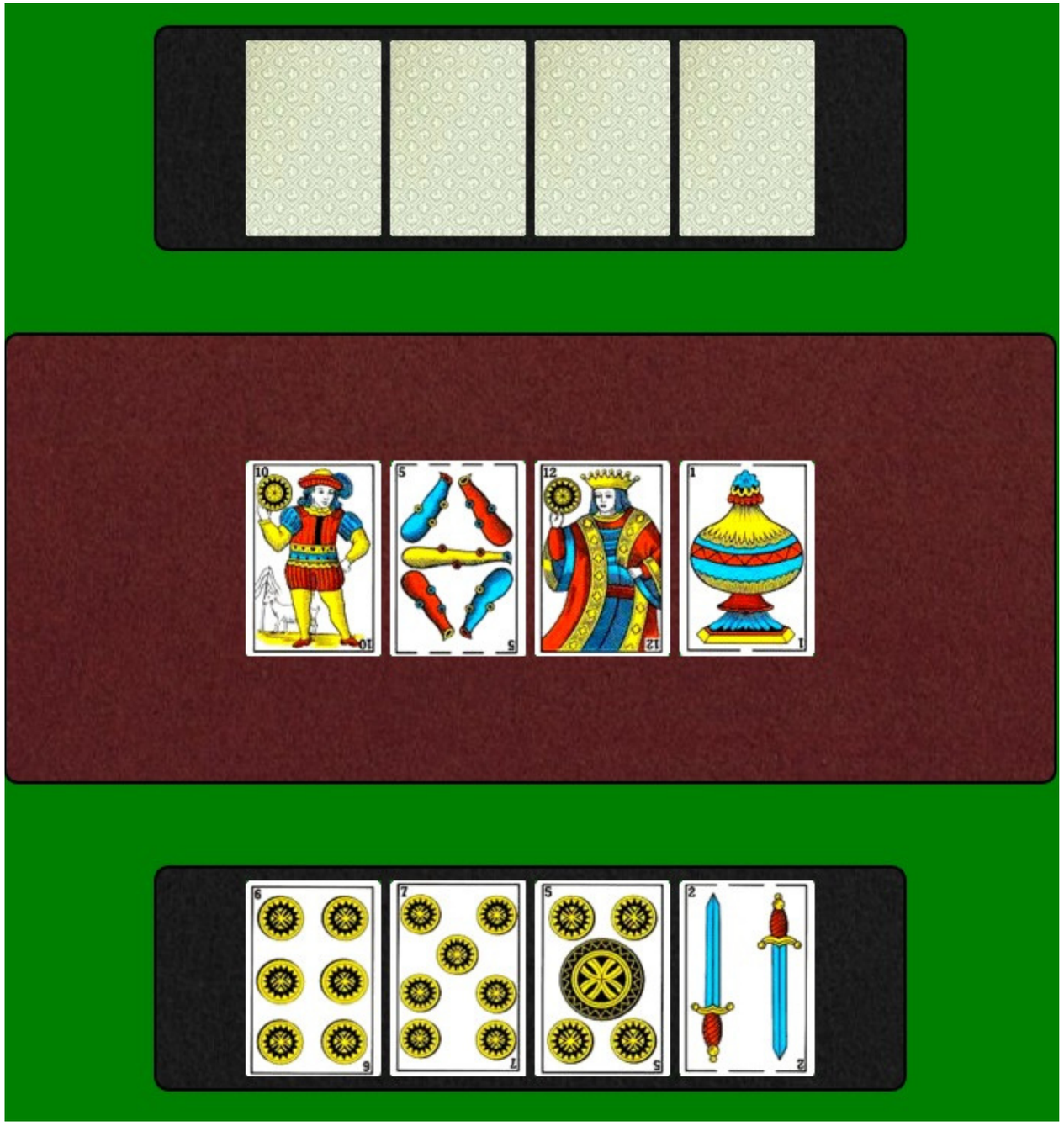
stock des cartes
clique pour
distribuer les
cartes

l’adversaire

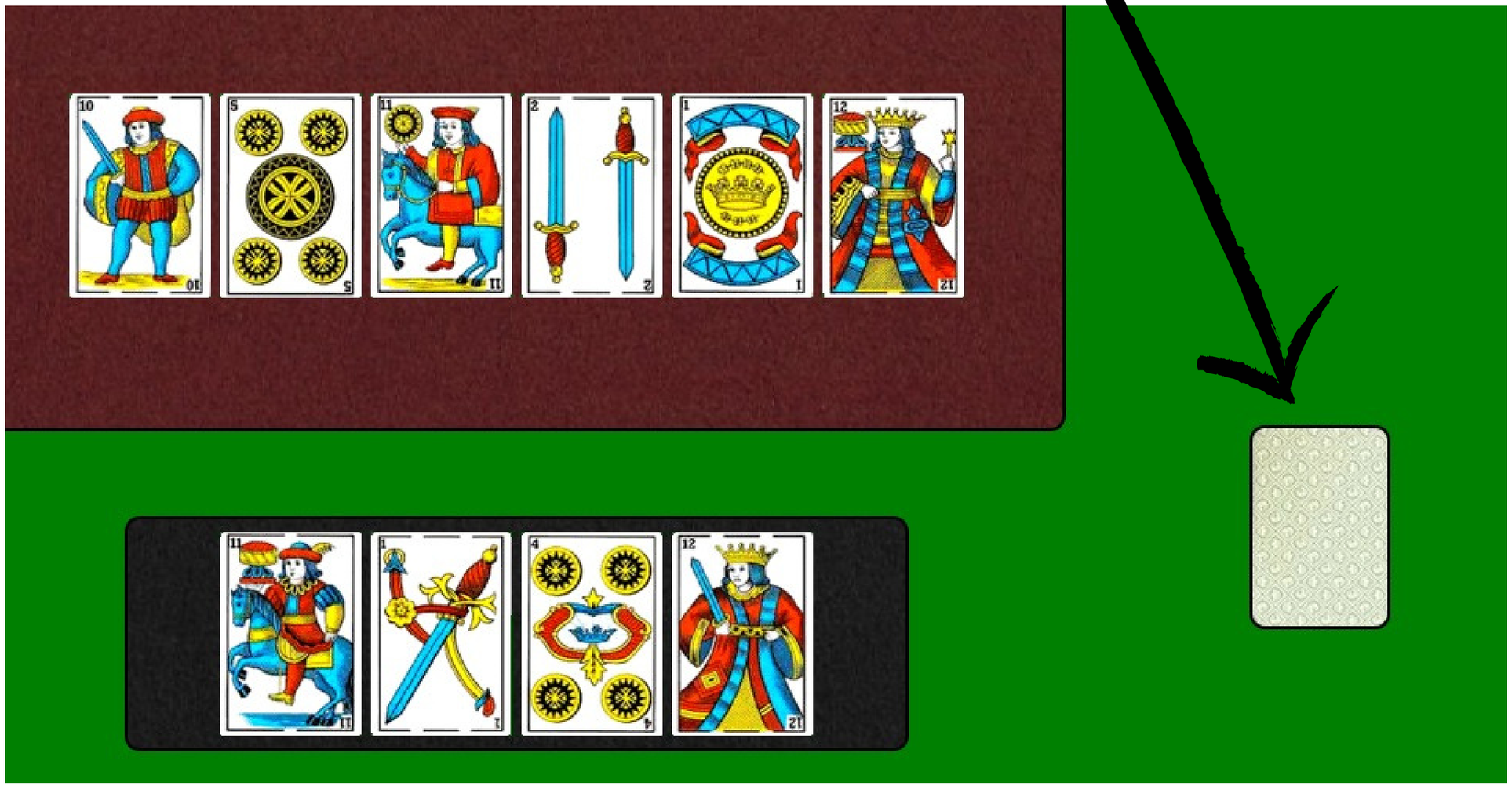
les certes a jouer

le user





les score des cartes



les tâches réalisées par chacun en cas de binôme.

Dans le cadre de leur projet de développement d'un jeu de cartes marocain, le binôme constitué de Yassine et d'Assaad a adopté une approche collaborative où les rôles se sont répartis de manière complémentaire. Yassine s'est principalement investi dans la conception et la création des frames du jeu, mettant en œuvre l'interface utilisateur à l'aide de la bibliothèque Qt. Son travail a inclus la configuration des différentes fenêtres et éléments visuels, avec un accent particulier sur la disposition esthétique des cartes sur les frames. De son côté, Assaad s'est concentré sur la partie algorithmique du projet, développant les classes et les algorithmes essentiels pour la logique de jeu. Cela englobait la gestion des règles du jeu, la manipulation des cartes, ainsi que la mise en place des mécanismes de prise de décision pour les joueurs virtuels. La collaboration entre Yassine et Assaad a permis une division efficace des tâches, capitalisant sur les compétences respectives pour créer une base solide et fonctionnelle pour leur jeu de cartes marocain.

Préparée par:

Nom : JARRADI
Prénom : YASSINE
CNE : F139074517
Email istitutionnel : yassine.jarradi@etu.uae.ac.ma

Nom : EL AOUZI
Prénom : ASSAAD
CNE : P149017686
Email istitutionnel : Assaad.elaouzi@etu.uae.ac.ma