



كلية العلوم والتكنولوجيات. كلية

RONDA

JEU DE CARTES MAROCAIN



DEVELOPE PAR : AYA KHALIFI

ENCARDE PAR PROF : IKRAM BEN ABDELOUAHAB

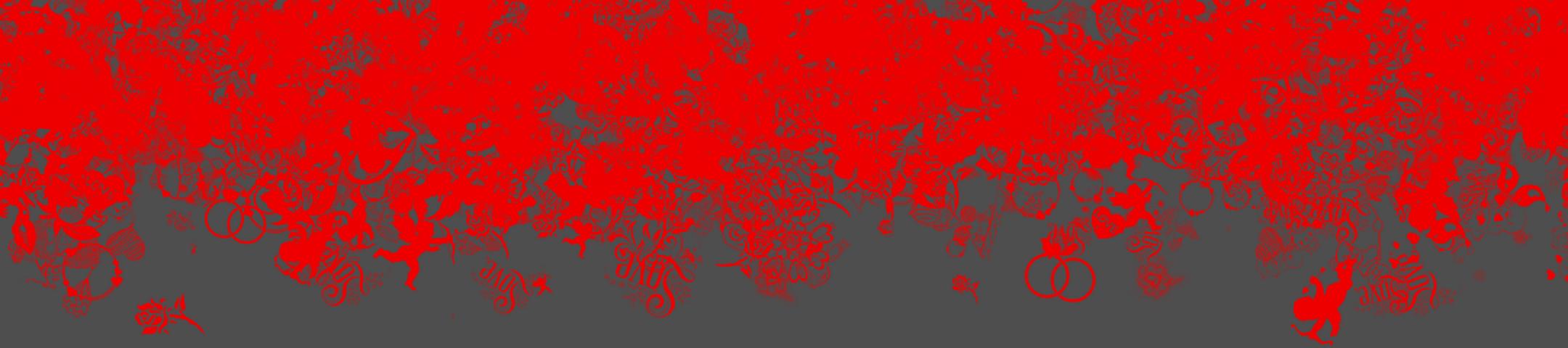


INTRODUCTION

Ce rapport documente le processus de conception et de développement d'un jeu de cartes RONDA interactif en utilisant le langage de programmation C++ et le framework Qt 6.

C++:

Ce projet se base sur le langage C++ en raison de sa performance, de sa flexibilité et de son adoption généralisée dans le développement logiciel. Sa capacité à combiner la programmation procédurale et orientée objet nous permet d'adopter une approche polyvalente pour créer des applications solides.



ARCHITECTURE DU CODE

QT 6:

Le framework Qt joue un rôle essentiel dans mon projet en fournissant une architecture robuste pour le développement d'interfaces utilisateur modernes et la gestion des événements. Sa capacité à fonctionner sans problème sur plusieurs plateformes en fait un choix idéal pour créer des applications multiplateformes.

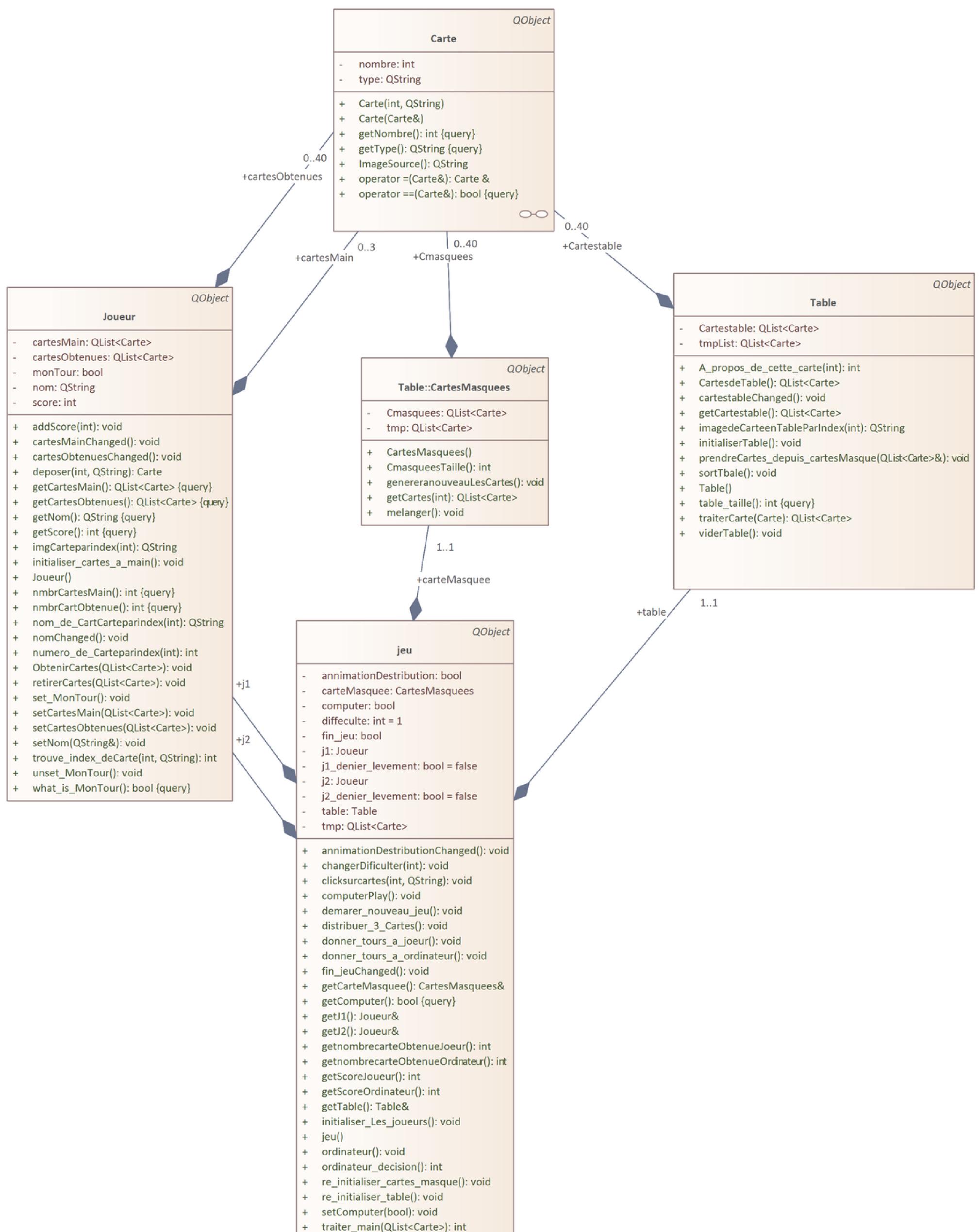
L'utilisation combinée de C++ et QML dans une application Qt crée une synergie puissante. Les applications Qt peuvent être conçues de manière à ce que la logique métier complexe soit écrite en C++, tandis que l'interface utilisateur est définie en QML. C++ et QML travaillent harmonieusement ensemble grâce au système de signaux de Qt.

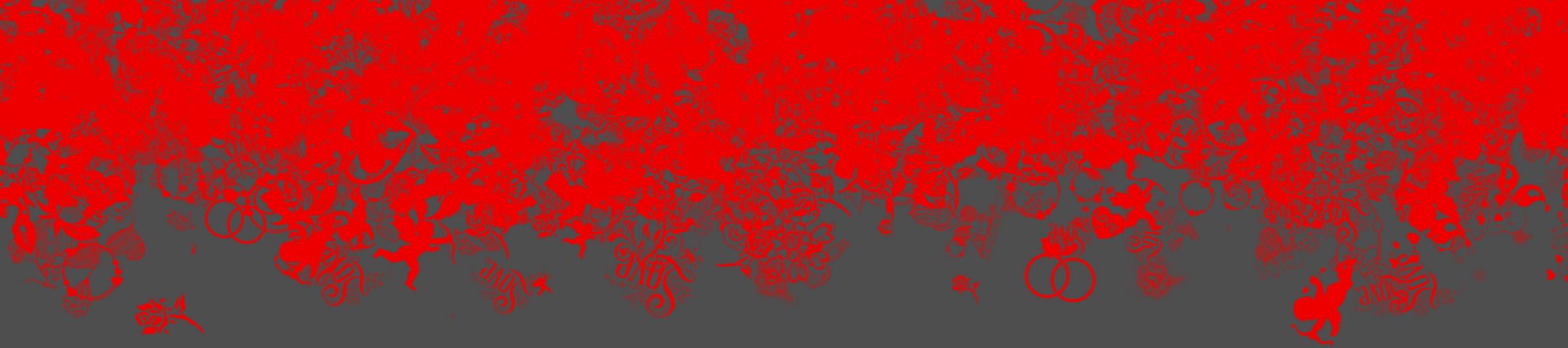
Communication de C++ vers QML : Les objets C++ peuvent être exposés dans QML sous forme de types personnalisés, permettant ainsi à la logique C++ d'être utilisée dans l'interface utilisateur QML.

Communication de QML vers C++ : Les signaux émis par QML peuvent être connectés aux slots C++, ce qui permet d'effectuer des actions côté C++ en réponse aux événements survenant dans le monde du QML.

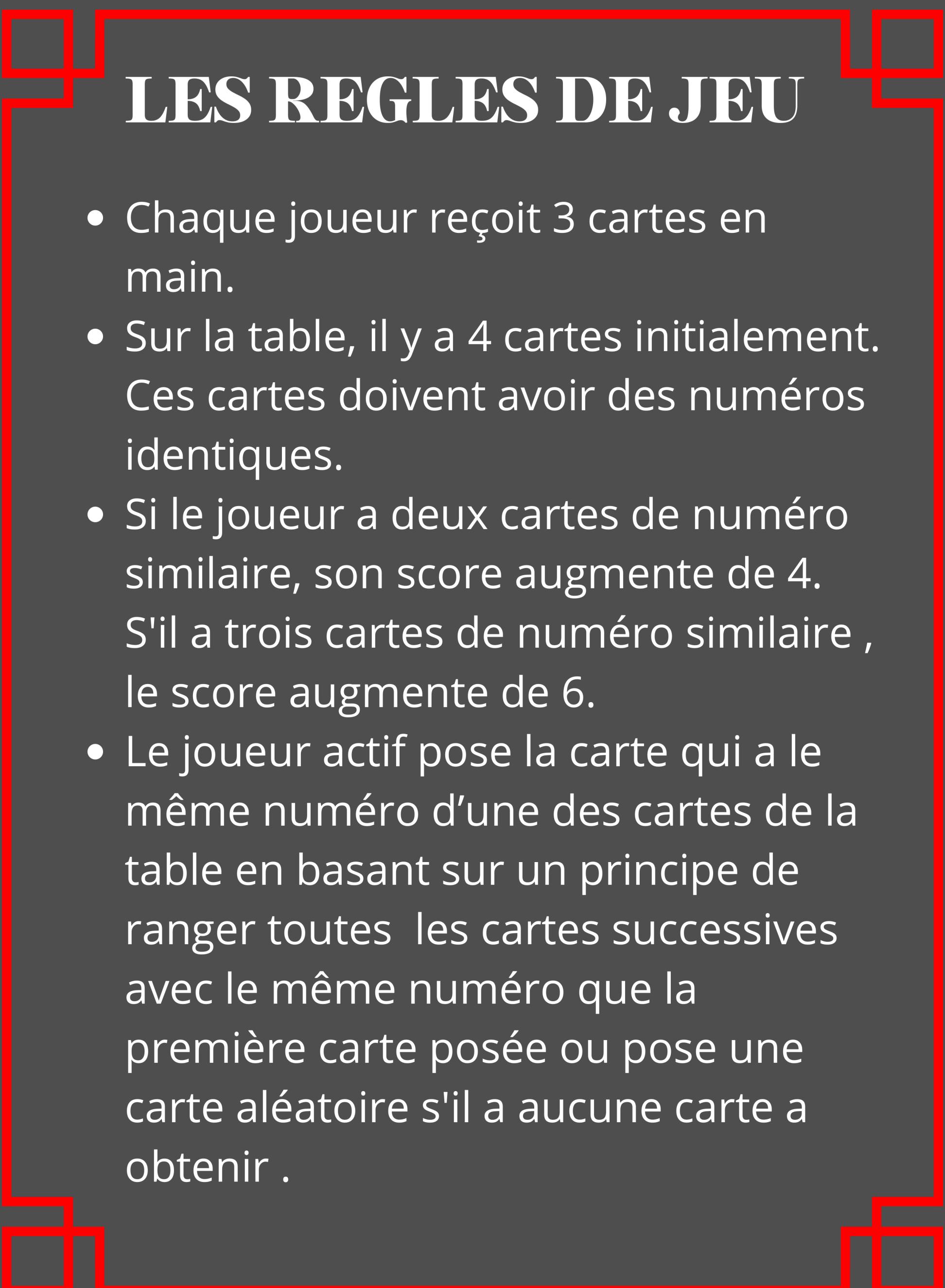
CONCEPTION DU JEU

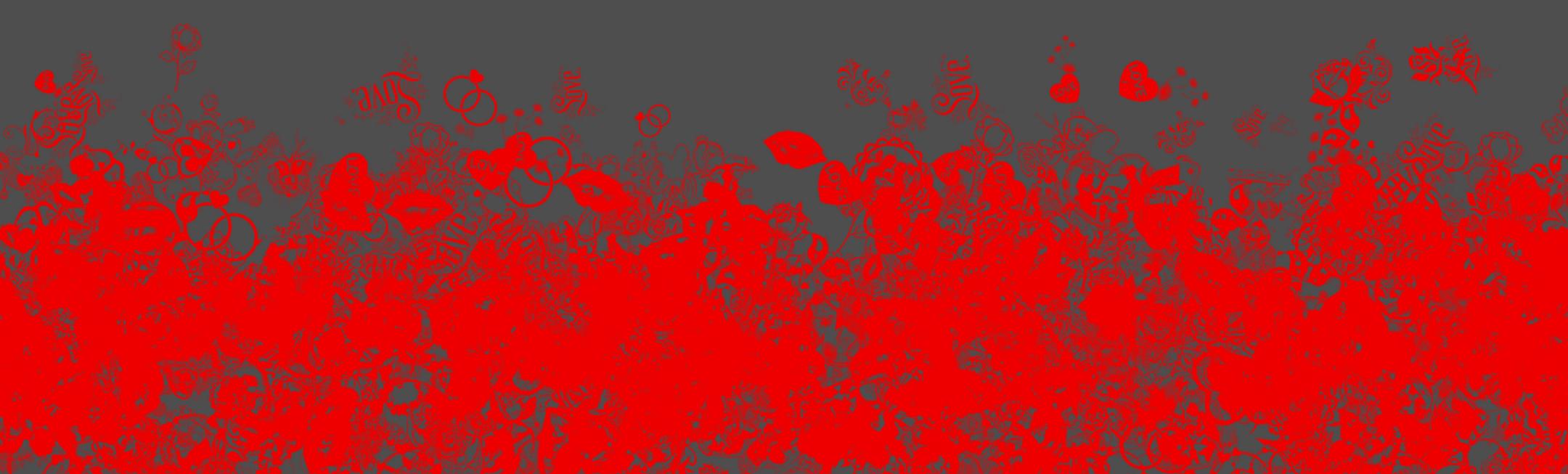
DIAGRAMME DE CLASSE





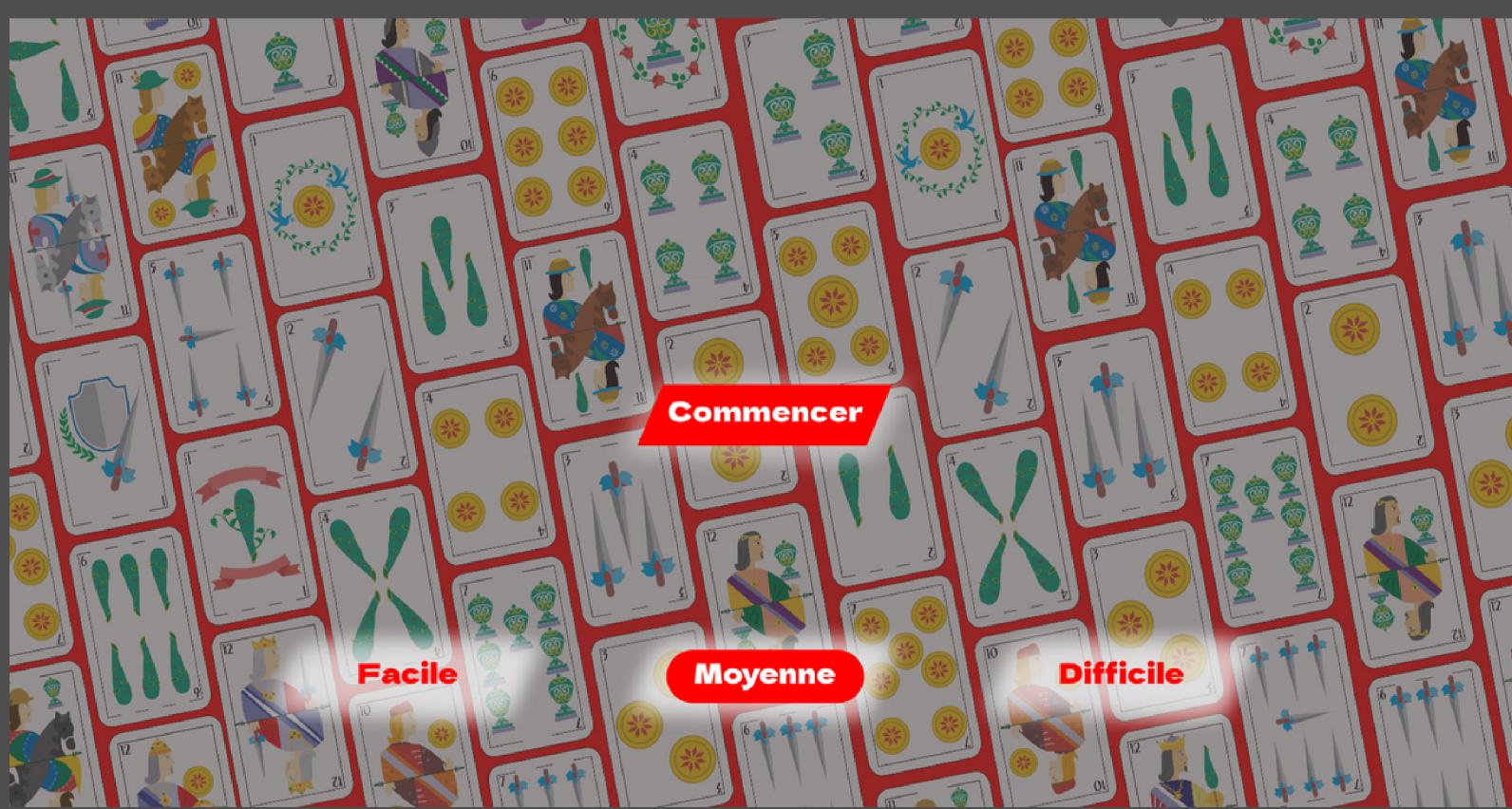
LES REGLES DE JEU



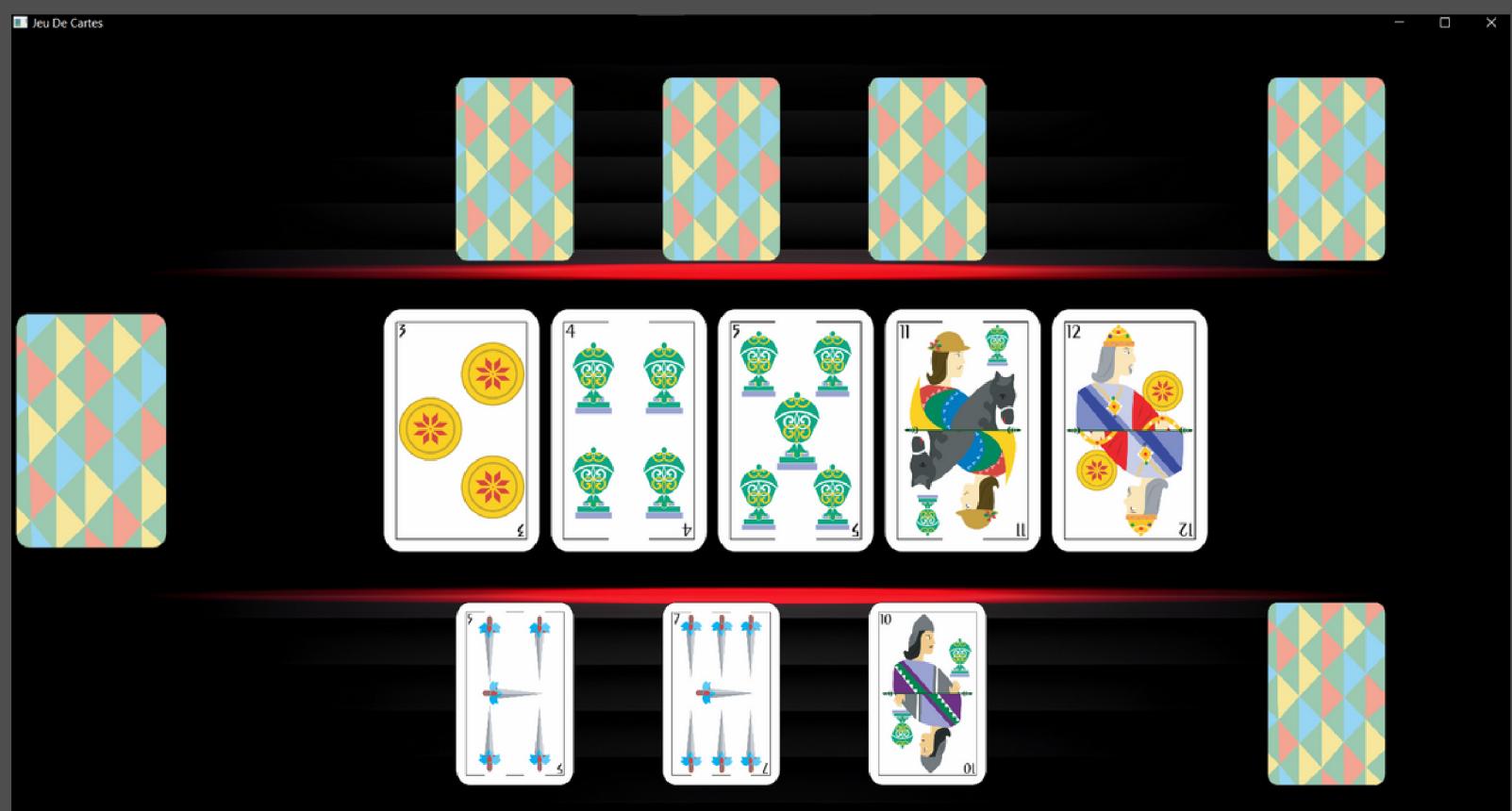
- Chaque joueur reçoit 3 cartes en main.
 - Sur la table, il y a 4 cartes initialement. Ces cartes doivent avoir des numéros identiques.
 - Si le joueur a deux cartes de numéro similaire, son score augmente de 4. S'il a trois cartes de numéro similaire , le score augmente de 6.
 - Le joueur actif pose la carte qui a le même numéro d'une des cartes de la table en basant sur un principe de ranger toutes les cartes successives avec le même numéro que la première carte posée ou pose une carte aléatoire s'il a aucune carte à obtenir .
- 

- Si un joueur réussit à ranger toutes ses cartes sur la table, son score augmente de 5 .
- Le nombre de cartes obtenues de joueur est ajouté au score obtenu.
- Une fois que le joueur pose ses trois cartes, il reçoit encore une fois trois cartes jusqu'a la fin de 6 tours .
- Le jeu continue en posant des cartes successives et essayant de ranger autant de cartes que possible. Le joueur avec le plus grand score gagne la partie.

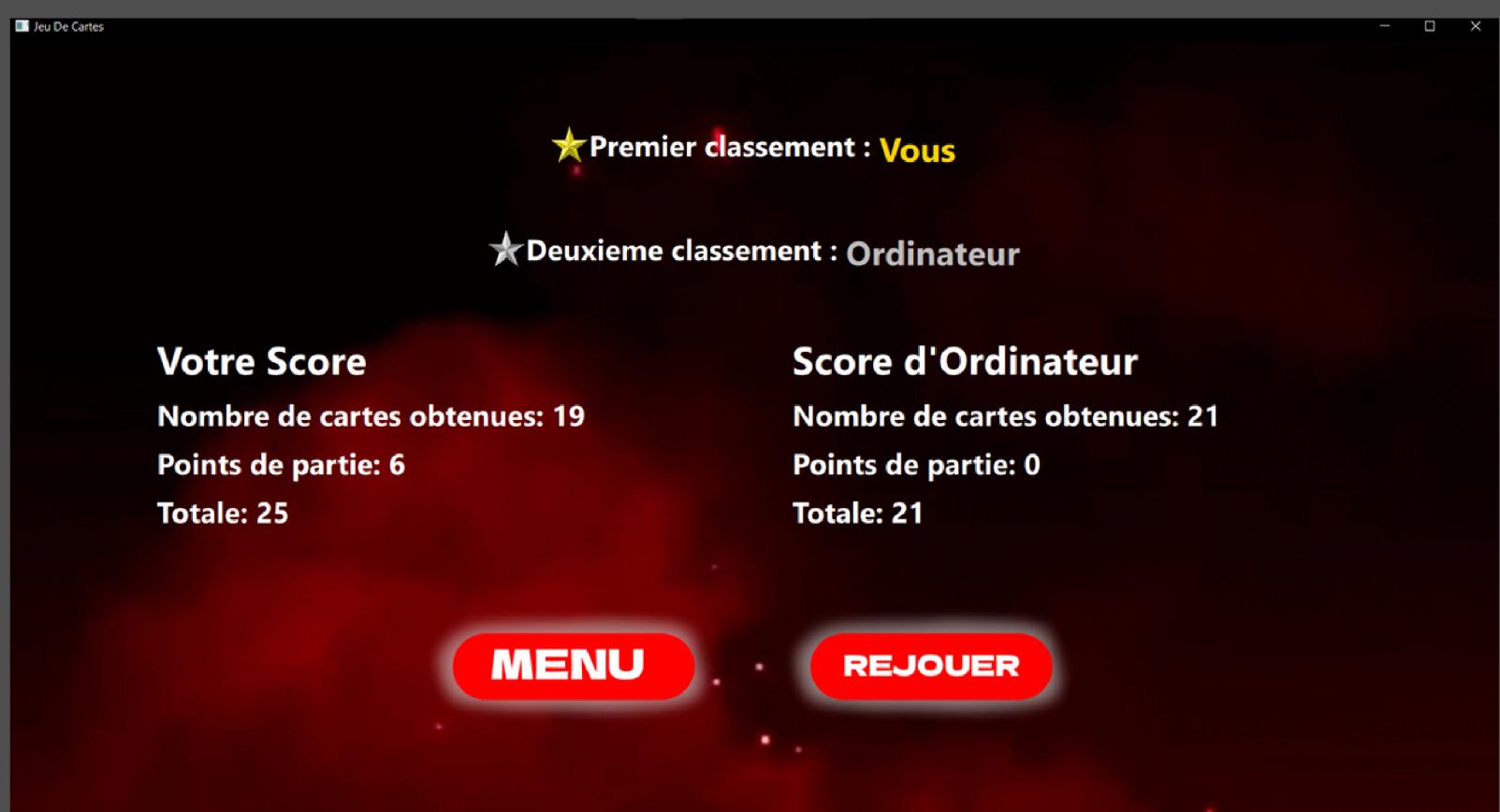
APERCU VISUEL DU JEU



Ecran d'accueil



Interface du jeu



Ecran de la fin du jeu

```
void jeu::demarer_nouveau_jeu(){
    re_initialiser_cartes_masque();
    re_initialiser_table();
    initialiser_Les_joueurs();
    // ici je fait signal a qml pour generer le view , puis j attend le click d utilisateur
    tmp.clear();

    emit animationDistributionChanged();

    // distribuer les cartes au joueurs et a table
    tmp = carteMasquee.getCartes(3);

    j1.retirerCartes(tmp);
    j1.set_MonTour();
    tmp.clear();

    tmp = carteMasquee.getCartes(3);

    j2.retirerCartes(tmp);
    j2.unset_MonTour();
    tmp.clear();

    tmp = carteMasquee.getCartes(4);
    table.prendreCartes_depuis_cartesMasque(tmp); /*prendre list de carte de cartes masquée*/
    tmp.clear();
```

Cette fonction est la fonction principale qui gère le début du jeu . elle sert à initialiser les éléments du jeu (cartes masquées, table, joueur et la liste temporaire qui prend les cartes).

Puis , elle fait appel a la fonction de distribution des cartes (3 cartes pour joueurs et 4 cartes pour la table) .

```
QList<Carte> CartesMasquees::getCartes(int n){  
    tmp.clear();  
  
    if(n == 4){  
  
        int var=0;  
        for (const Carte c: Cmasquees){  
  
            //  
            auto it = std::find(tmp.begin(), tmp.end(), c);  
  
            auto it = std::find_if(tmp.begin(), tmp.end(), [c](const Carte& tmpCard) {  
                return tmpCard.getNombre() == c.getNombre();  
            });  
  
            if (it == tmp.end()) {  
                tmp.append(c);  
                Cmasquees.removeOne(c);  
                var++;  
                if(var == 4){  
                    return tmp;  
                    break;  
                }  
            }  
        }  
    }  
}
```

cette partie de fonction verifie si les cartes de tables sont similaires par numero , si oui elle remplace la carte par une autre de cartes masquees.

```

else{

    for (int i = 0; i < n; ++i) {
        Carte tmpCarte(Cmasquees.at(0));
        tmp.append(tmpCarte);
        Cmasquees.removeAt(0);
    }
}

return tmp;

```

Si non cette fonction ajoute les cartes sur table et les supprimer de cartes masquees

```

void jeu::distribuer_3_Cartes(){

    if(carteMasquee.CmasqueesTaille() != 0){
        emit animationDestributionChanged();

        tmp.clear();
        tmp = carteMasquee.getCartes(3);/***/
        j1.retirerCartes(tmp);
        j1.set_MonTour();

        /* ici je check si il a ronda ou ronda de rondes */
        int xpj1 = traiter_main(tmp);
        j1.addScore(xpj1);

        tmp.clear();
        tmp = carteMasquee.getCartes(3);/***/
        j2.retirerCartes(tmp);
        j2.unset_MonTour();

        /* ici je check si il a ronda ou ronda de rondes */
        int xpj2 = traiter_main(tmp);
        j2.addScore(xpj2);
    }
}

```

Cette fonction permet de distribuer les 3 cartes pour les joueurs quand ils posent toutes les cartes jusqu'au le packet de cartes masquees soit vide. elle traite les cartes a main des joueurs pour verifier s'il a deux ou trois cartes similaires de numero a fin d'augmenter le score .

```
else if (carteMasquee.CmasqueesTaille() == 0){

    if ( j1_denier_levement ){
        //donner au j1 les cartes chaytin f table dans dernier tous
        j1.ObtenirCartes(table.getCartestable());
        table.viderTable();
    }

    else if(j2_denier_levement){
        //donner au j2 les cartes chaytin f table dans dernier tous
        j2.ObtenirCartes(table.getCartestable());
        table.viderTable();
    }

    // emit signal to finish game
    emit fin_jeuChanged();
    return;
}
```

Si les cartes masquees sont vide (fin du jeu) , elle verifie le dernier joueur qui a obtenu des cartes de la table pour le donner les cartes restantes sur la table.

```
void jeu::clicksurcartes(int carte_nombre,QString carte_nom){  
    if (j1.what_is_MonTour()){  
  
        // pass it to table , and let it return qlist , either filled or empty , doesnt matter  
        Carte tmp(j1.deposer(carte_nombre,carte_nom));  
  
        QList t =  table.traiterCarte(tmp);  
  
        if (t.size() != 0){  
            /*im the last player lifted from table*/  
            j1_denier_levement = true;  
            j2_denier_levement = false;  
        }  
  
        j1.ObtenirCartes(t);  
  
        // savoir combien de cartes dans la table , si missa je gagne 5 XP  
        if (table.table_taille() == 0){  
            j1.addScore(5);  
        }  
    }  
  
    // switch turns  
    donner_tours_a_ordinateur();
```

Cette fonction verifie le tour de joueur pour le permettre de deposer une carte . Si non il ne depose pas , et traiter la carte depose (cette fonction de traitement cherche les cartes successives de carte deposee pour les ajouter aux cartes obtenues _) et donner le tour apres a l'ordinateur .

```
QList<Carte> Table::traiterCarte(Carte cartePoseParJoueur){  
    /* 1- chercher ila kayna chi carta mathez */  
    int numeroPoserParjoueur = cartePoseParJoueur.getNombre();  
  
    QList<Carte> t;  
  
    int nbrCartesLevees=0;  
    int Ndepart=-1;  
  
    // tableau contient les numero des cartes de jeu  
    QList<int> tableauNum= { 1,2,3,4,5,6,7,10,11,12 };  
  
    // on doit savoir l index dans le tableau precedent , le numero de la carte de  
    int index = tableauNum.indexOf(numeroPoserParjoueur);  
  
    // we loop to find first elemet to start poping from  
    for(int v =0 ; v < Cartestable.size() ; ++v){  
        if(Cartestable[v].getNombre() ==  tableauNum[index] ){  
            Ndepart = v;  
            break;  
    }  
}
```

```
void jeu::ordinateur(){  
  
    if( j2.what_is_MonTour() ){  
  
        int index_de_carte_a_poser = ordinateur_decision(); // index_de_carte_a_poser  
  
        int carte_nombre = j2.numero_de_Carteparindex(index_de_carte_a_poser);  
        QString carte_nom = j2.nom_de_CartCarteparindex(index_de_carte_a_poser);  
  
        QList t = table.traiterCarte( j2.deposer(carte_nombre,carte_nom) );  
  
        if (t.size() != 0){  
            /*im the last player lifted from table*/  
            j1_denier_levement = false;  
            j2_denier_levement = true;  
        }  
  
        // j2 leve les cartes obtenues  
        j2.ObtenirCartes(t);  
  
        // savoir combien de cartes dans la table , si missa je gagne 5 XP  
        if (table.table_taille() == 0){  
            j2.addScore(5);  
        }  
  
        donner_tours_a_joueur();  
    }  
}
```

la fonction ordinateur gere ses operations apres la definition de son tour et fait appel a la fonction qui gere la decision d'ordinateur selon la difficulte choisi.

```

int ordinateur_decision(){
    // pour chaque carte dans la main
    // il doit savoir la valeur de chaque carte dans la table
    // selon la difficulté va retourner l'index de la carte que l'ordinateur doit déposer
    int carte_max_potentiel = -1, carte_min_potentiel = 999;

    for (int i = 0;i < j2.nbrCartesMain();i++){

        int potentiel =  table.A_propos_de_cette_carte( j2.numero_de_Carteparindex(i));

        if(potentiel > carte_max_potentiel ){
            carte_max_potentiel = i;
        }
        if (potentiel < carte_min_potentiel ){
            carte_min_potentiel = i;
        }

    }

    if(difficulte == 3 /* difficile: on prend max potentiel*/){
        return carte_max_potentiel;

    }else if ( difficulte == 1){
        return carte_min_potentiel;

    }else if (difficulte == 2){
        /*randomly return either max or min*/
        bool choose = QRandomGenerator::global()->bounded(2) == 1;
        return choose ? carte_max_potentiel : carte_min_potentiel;
    }
}

```

cette fonction compare toutes les cartes à main de l'ordinateur, si difficulté est facile il pose la carte qui le moins potentiel si difficulté est moyen ,

il pose une carte aleatoire. Si difficile, il pose une carte qui a le plus grand potentiel entre les cartes et tous en basant sur fonction "a propos de carte" qui verifie la carte a main qui peut gagner le plus grand nombre du cartes de la table.

```
int A_propos_de_cette_carte(int x){  
    // this function should return how many cards i can obtain if i chose to put card i  
    // it will be used inorder for computer use it accordingly to defficulty  
  
    // tableau contient les numero des cartes de jeu  
    QList<int> tableauNum= { 1,2,3,4,5,6,7,10,11,12 };  
  
    int nbrCartesLevees=0;  
    int Ndepart=-1;  
  
    // on doit savoir l index dans le tableau precedent , le numero de la carte deposee  
    int index = tableauNum.indexOf(x);  
  
    // we loop to find first elemet to start poping from  
    for(int v =0 ; v < Cartestable.size() ; ++v){  
        if(Cartestable[v].getNombre() ==  tableauNum[index] ){  
            Ndepart = v;  
            break;  
        }  
    }  
    // here we loop to find how many cards to pop after the starting point  
    if(Ndepart != -1){  
        for (int var = 0; var < Cartestable.size() ; ++var) {  
            Carte tm(Cartestable[var]);  
            int n= tableauNum[index];
```

```
QQmlApplicationEngine engine;  
  
jeu j;  
  
engine.rootContext()->setContextProperty("jeu",&j);  
engine.rootContext()->setContextProperty("j1",&(j.getJ1()));  
engine.rootContext()->setContextProperty("j2",&(j.getJ2()));  
engine.rootContext()->setContextProperty("Table",&(j.getTable()));  
engine.rootContext()->setContextProperty("cartesMasquees",&(j.getCarteMasquee()));
```

ici dans main on expose les instances de chaque classe dans qml engine pour qu'il puisse entendre le signal venant de c++.

Ce que j'ai acquis:

- Integration C++ et QML : Experience dans l'integration transparente entre le code C++ et l'interface utilisateur QML a l'aide du framework Qt.
- Utilisation des Signaux et Slots : Maîtrise de la communication entre les composants a l'aide du systeme de signaux et slots de Qt.
- Conception QML : Conception d'interfaces utilisateur interactives et attrayantes en utilisant QML.

RESSOURCES

Qt Documentation

<https://doc.qt.io/>

Qt Academy

<https://academy.qt.io/>

Qt QML Book

<https://www.qt.io/product/qt6/qml-book>

Youtube : VoidRealms

[https://youtu.be/7aonJh5f_YA?
si=pqXn0dShB00W7w0E](https://youtu.be/7aonJh5f_YA?si=pqXn0dShB00W7w0E)

pouya azimi

[https://youtu.be/Nma3c3YxsUo?
si=rPbRsLSfLQ142uGZ](https://youtu.be/Nma3c3YxsUo?si=rPbRsLSfLQ142uGZ)

Marko uses Qt

[https://youtu.be/-7zNtRabCiU?
si=ZILinioJNXKzKpNi](https://youtu.be/-7zNtRabCiU?si=ZILinioJNXKzKpNi)