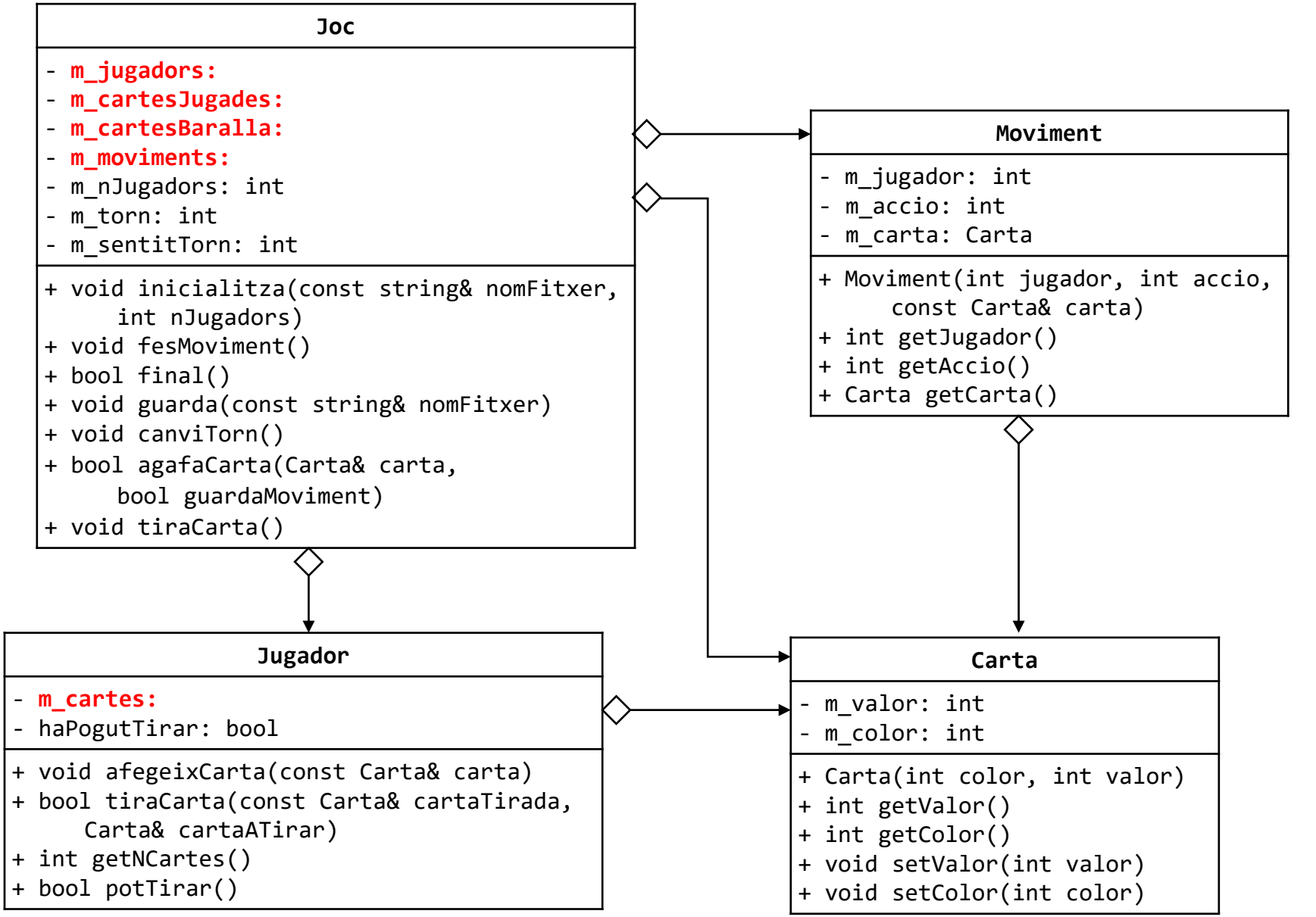
Tema 2 – Exercici Opcional 4

Volem fer un **programa** que ens simuli una partida d'un **joc de cartes** inspirat en el popular joc del "**Uno**". En aquest joc hi poden participar de **2 a 4 jugadors**. Inicialment **cada jugador té 7 cartes** i **l'objectiu** final del joc és **descartar-se de totes les cartes**. Les **cartes** tenen un **color** (vermell, groc, verd o blau) i un **valor del 0 al 9**. A més a més, per cada color hi ha diferents **cartes especials**. Nosaltres només considerarem 3 tipus de cartes especials: la carta que fa que el següent jugador hagi de **robar** obligatòriament **dues cartes** abans de poder tirar, la carta que fa **canviar el sentit del joc**, de forma que torna a jugar el jugador anterior, i la carta que fa **saltar el torn**, de forma que el següent jugador perd el seu torn de joc.

Quan un jugador té el torn pot **tirar una carta del mateix color o del mateix valor** que l'última carta jugada pel jugador anterior, **o una carta especial del mateix color**. Si no pot tirar cap carta haurà de robar tantes cartes de la baralla fins que pugui tirar o s'acabin les cartes de la baralla. Guanya el primer jugador que es queda sense cartes. En la nostra versió, si la partida arriba a una situació en què cap jugador pot tirar s'acaba amb empat.

Tema 2 – Exercici Opcional 4



Tema 2 – Exercici Opcional 4

Classe `Carta`: guarda la informació d'una carta del joc (color i valor). Per indicar el valor de les cartes especials hi ha definides dins del codi que us donem les constants numèriques `CANVI_TORN`, `ROBA_DOS` i `SALTA_TORN`. Per codificar el color també teniu definides les constants `VERMELL`, `VERD`, `BLAU` i `GROC`.

Carta
- m_valor: int - m_color: int
+ Carta(int color, int valor) + int getValor() + int getColor() + void setValor(int valor) + void setColor(int color)

Classe `Jugador`: guarda i gestiona la informació de cadascun dels jugadors de la partida. Té dos atributs: `m_cartes`, per guardar totes les cartes que té el jugador en cada moment de la partida i el booleà `m_haPogutTirar` que ens indica si el jugador ha pogut tirar carta al seu últim torn o no. Aquesta informació ens servirà per poder decidir si la partida s'acaba perquè cap jugador pot tirar.

Jugador
- m_cartes: - haPogutTirar: bool
+ void afegeixCarta(const Carta& carta) + bool tiraCarta(const Carta& cartaTirada, Carta& cartaATirar) + int getNCartes() + bool potTirar()

Classe `Moviment`: guarda la informació de tots els moviments que van fent cadascun dels jugadors. Ens servirà per poder verificar el resultat de la partida amb el test automàtic d'avaluació. De cada moviment que es fa guardem el número de jugador que l'ha fet, el tipus de moviment (robar o tirar carta) i la carta que ha tirat o ha robat.

Moviment
- m_jugador: int - m_accio: int - m_carta: Carta
+ Moviment(int jugador, int accio, const Carta& carta) + int getJugador() + int getAccio() + Carta getCarta()

Tema 2 – Exercici Opcional 4

Classe `Joc`: aquesta classe guarda tota la informació necessària per gestionar una partida del joc. Té els atributs següents:

- `m_jugadors`, amb la informació de tots els jugadors que participen a la partida.
- `m_cartesJugades`, amb les cartes que els jugadors han tirat durant la partida.
- `m_cartesBaralla`, amb les cartes de la baralla que encara no s'han repartit entre els jugadors. Cada cop que un jugador ha de robar agafa la carta de dalt de la baralla.
- `m_moviments`, per guardar tots els moviments que es fan a la partida en l'ordre que es van fent.
- `m_nJugadors:` El número de jugadors.
- `m_torn:` Quin jugador té el torn i per tant, ha de fer el següent moviment.
- `m_sentitTorn:` el sentit del canvi de torn que indica si al passar el torn s'incrementa o redueix el número de jugador.

Joc
- **m_jugadors:** - **m_cartesJugades:** - **m_cartesBaralla:** - **m_moviments:** - m_nJugadors: int - m_torn: int - m_sentitTorn: int
+ void inicialitza(const string& nomFitxer, int nJugadors) + void fesMoviment() + bool final() + void guarda(const string& nomFitxer) + void canviTorn() + bool agafaCarta(Carta& carta, bool guardaMoviment) + void tiraCarta()

Tema 2 – Exercici Opcional 4

- **Penseu quina estructura de dades** de les que hem explicat a classe (**llista, pila, cua, vector**) és més adequada per guardar l'atribut **`m_cartes`** de la classe **`Jugador`** i els atributs **`m_jugadors`**, **`m_cartesJugades`**, **`m_cartesBaralla`** i **`m_moviments`** de la classe **Joc**.

  Per cadascun d'ells penseu quina de les classes de la llibreria estàndard (**`vector`, `forward_list`, `list`, `stack`, `queue`**) hem d'utilitzar.

Tema 2 – Exercici Opcional 4

A **nivell de mètodes**, us donem **ja implementats** els mètodes bàsics, `getters` i `setters`, de les **classes `Carta` i `Moviment`**.

Per la **classe `Jugador`** heu d'**implementar** aquests dos **mètodes**:

- **`afegeixCarta`**: afegeix una carta al conjunt de cartes del jugador.
- **`tiraCarta`**: aquest mètode ha de seleccionar una de les cartes de les que té el jugador per poder tirar en el seu torn. En el paràmetre `cartaTirada` rep la carta que ha tirat el jugador anterior i que està a dalt de tot de les cartes jugades. Si té alguna carta que es pugui tirar segons les regles del joc, la retornarà al paràmetre per referència `cartaATirar` i retornarà `true` com a resultat de la funció. Si no té cap carta per tirar retornarà `false`. En qualsevol dels dos casos s'ha d'actualitzar l'atribut que indica si el jugador ha pogut tirar o no en el seu torn.

  Si el jugador té més d'una carta que es pot tirar, la carta a tirar es seleccionarà seguint aquest ordre de prioritat:

  1. Carta especial de robar 2 del color de l'última carta tirada.
  2. Carta especial de canvi de torn del color de l'última carta tirada.
  3. Carta especial de saltar torn del color de l'última carta tirada.
  4. Una carta normal del mateix color de l'última carta tirada. Si en tenim més d'una, tirarem la que tingui el valor més petit.
  5. Una carta normal del mateix valor de l'última carta tirada, però de diferent color. Si en tenim més d'una tirarem la que tingui el codi de color més petit segons les constants de color declarades al fitxer "`Carta.h`".

Tema 2 – Exercici Opcional 4

Per la **classe `Joc`** us donem **implementats** pràcticament de forma completa els **mètodes** següents (només haureu de completar la part del codi que s'indica en cada cas):

- **`inicialitza`**: llegeix d'un fitxer totes les cartes del joc en un determinat ordre i reparteix les 7 cartes inicials a cada jugador. La resta de cartes queden col·locades a la baralla, menys la última que es posa coma primera carta jugada perquè sigui la carta a partir de la que comença el joc. També s'inicialitza el torn al primer jugador. Les cartes s'han de repartir als jugadors en ordre invers a com s'han llegit del fitxer.

  En aquest mètode **haureu de completar** allà on s'indica amb comentaris dins del codi les instruccions necessàries per afegir una carta a la baralla utilitzant l'atribut `m_cartesBaralla`, afegir un jugador al conjunt de jugadors utilitzant l'atribut `m_jugadors` i afegir la primera carta a les cartes jugades amb l'atribut `m_cartesJugades`. Haureu de posar el codi que calgui **en funció de l'estructura de dades** que hagueu escollit per cada atribut.
- **`fesMoviment`**: decideix què s'ha de fer en funció de l'última carta tirada. Si és una carta especial, es fa primer l'acció que correspon a la carta especial (es canvia el sentit del torn, es salta un torn o es força al jugador que té el torn a robar dues cartes). Després fa que el jugador que té el torn tiri una carta i finalment es passa el torn al següent jugador.

  En aquest mètode **haureu de completar** allà on s'indica amb comentaris dins del codi la instrucció necessària per recuperar l'última carta jugada utilitzant l'atribut `m_cartesJugades`, **en funció de l'estructura de dades** que hagueu escollit per aquest atribut.
- **`canviTorn`**: passa el torn al següent jugador segons el sentit del joc.

Tema 2 – Exercici Opcional 4

A la classe **Joc heu d'implementar** els mètodes següents:

- **`agafaCarta`**: aquest mètode es cridarà cada cop que un jugador hagi de robar una carta de la baralla. Ha de treure una carta de la baralla i retornar-la al paràmetre per referència `carta`. Si la baralla està buida ha de retornar `false`, i `true` en cas contrari. A més a més, si el paràmetre `guardaMoviment` està a `true` s'ha de guardar la informació del moviment que s'ha fet (nº de jugador, ROBA_CARTA i el valor de la carta) a l'estructura que guarda tots els moviments de la partida.
- **`tiraCarta`**: aquest mètode es cridarà quan un jugador hagi de tirar una carta. Ha de cridar al mètode `tiraCarta` del jugador que té el torn. Si el jugador no té cap carta per tirar ha de robar una carta de la baralla, afegir-la a les cartes del jugador i tornar a intentar tirar una carta, repetint fins que o bé pugui tirar carta o ja no quedin cartes per robar. Ha de guardar tots els moviments que es facin (tant els de robar com els de tirar carta).
- **`final`**: determina si s'ha arribat al final de la partida. Hi ha dos casos en què s'arriba al final de la partida: si algun jugador es queda sense cartes o si cap dels jugadors ha pogut tirar en el seu últim torn de joc.
- **`guarda`**: guarda tots els moviments que s'han guardat en el fitxer que es passa com a paràmetre, seguint l'ordre en què s'han anat fent durant la partida. Cada moviment ha d'estar en una línia diferent del fitxer amb el nº de jugador, el codi del moviment (robar o tirar), el color de la carta i el valor de la carta, separats per espais en blanc.

El programa principal de prova que us donem simula el funcionament d'una partida a partir d'un fitxer amb l'ordre inicial de les cartes. Es van fent moviments fins que es detecta el final de la partida. Al final es guarden tots els moviments que s'han fet en un fitxer de sortida i aquest fitxer de sortida es compara amb el llistat de moviments esperat.