Gaming Platform – Beschreibung der Klassen

Für den Test im Moped sind genau folgende Dateien zu erzeugen:

game.h

game.cpp

player.h

player.cpp

gamekey.cpp

gamekey.h

Des Weiteren gibt es `enum class Mode{Ranked,Unranked};`.
Zum Test dürfen Sie nur die **Basisimplementierung** mitbringen. Speichern Sie sich also gegebenenfalls einen Zwischenstand bevor Sie die Erweiterung für den Zusatzpunkt implementieren. **Abgabe der Basisimplementierung bis 14.05.21**

1 Game

Die Klasse `Game` hat folgende **Instanzvariablen**.

string name Name des Spiels. (Zur Vereinfachung darf davon ausgegangen werden, dass die Namen der Game-Objekte eindeutig sind.)

weak_ptr<Player> host Leiter des Spiels.

map<string, weak_ptr<Player>> players Map von teilnehmenden `Player`-Objekten.

Die Klasse `Game` hat folgende **Konstrukturen und Methoden**.

Game(string name, shared_ptr<Player> host) Setzt Instanzvariablen. Name darf nicht leer und `host` nicht `nullptr` sein. Sollte ein Parameter nicht den vorgegebenen Werten entsprechen, ist eine Exception vom Typ `runtime_error` zu werfen.

string get_name() const Liefert den Namen des `this`-Objekts.

bool is_allowed(int n) const Liefert `true`, falls `n` größer als 90% und kleiner als 110% des MMRs von `host` ist, ansonsten `false`. (Hinweis: Um Rechenfehler mit double Werten zu vermeiden, sind die für die Vergleiche notwendigen Berechnungen komplett im int-Bereich durchzuführen.)

bool remove_player(const GameKey& gk, shared_ptr<Player> p) Entfernt `p` aus der Map der teilnehmenden Spieler, wenn möglich. Liefert `true`, falls `p` entfernt wurde, ansonsten `false`. GameKey siehe weiter unten.

bool add_player(const GameKey& gk, shared_ptr<Player> p) `p` soll zur Map der teilnehmenden Spieler hinzugefügt werden. Das ist nicht möglich, falls die Map das Spieler-Objekt bereits enthält oder das MMR von `p` um mehr als 10% vom MMR des Hosts abweicht (Zur Prüfung des MMR kann die Methode `is_allowed` verwendet werden). Es ist `true` zu retournieren, wenn das Einfügen erfolgreich war, `false` sonst. Begründung und Erklärung von GameKey siehe weiter unten.

size_t number_of_players() const Liefert Anzahl der aktiven Spieler (das sind alle Spieler-Objekte in der Map der teilnehmenden Spieler, die über den `weak_ptr` noch erreichbar sind, das heißt, `expired` liefert `false`).

weak_ptr<Player> best_player() const Enthält die Map der teilnehmenden Spieler keine aktiven Spieler (`number_of_players` liefert 0), so ist eine Exception zu werfen, die den Typ `runtime_error` hat. Ansonsten ist ein Pointer auf das teilnehmende `Player`-Objekt mit höchstem MMR zu liefern. Bei mehreren Objekten mit maximalem MMR, ist das erste in der Map auftretende zu liefern.

virtual int change(bool) const = 0 Pure virtual Methode. Die Methode retourniert den Wert, um den das MMR eines Spieler-Objekts zu ändern ist, wenn ein Spiel gewonnen (Parameterwert ist `true`) oder verloren (Parameterwert ist `false`) wurde.

weak_ptr<Player> play(size_t i) Im ersten Schritt ist die Map der teilnehmenden Spieler zu bereinigen. Das heißt, alle `weak_pointer`, für die `expired` `true` liefert, werden aus der Map entfernt. Sollte `i` anschließend nicht kleiner als die Anzahl der Einträge in der Map sein oder der Host nicht mehr existieren (`weak_ptr` `expired` liefert `true` bzw. `lock` schlägt fehl), ist eine Exception vom Typ `runtime_error` zu werfen. Das Spieler-Objekt mit Index `i` in der bereinigten Map wird zum Sieger deklariert. Alle anderen teilnehmenden Spieler-Objekte sind damit Verlierer. Das MMR aller Verlierer wird aktualisiert, indem der von `change(false)` gelieferte Wert dazu addiert wird (Methode `Player::change_mmr`). Bei Verlierern, deren ursprüngliches MMR größer als das MMR des Sieger-Objekts war, ist das doppelte des von `change(false)` gelieferten Wertes zu addieren. **Danach** wird der von `change(true)` gelieferte Wert zum MMR des Gewinner-Objekts addiert. Retourniert wird ein Pointer auf das Gewinner-Objekt.

virtual ~Game() = default Wegen Verwendung von Vererbungs-Polymorphismus notwendig

Die Klasse `Game` hat folgendes **Ausgabeformat**.

virtual ostream& print(ostream& o) const Gibt das Objekt auf den ostream `o` aus. Format: [`name`, `host->name`, `host->mmr`, player: {[`Player_name`, `Player_mmr`], [`Player_name`, `Player_mmr`], ... }]

operator<< `Game`-Objekte sollen zusätzlich über operator<< ausgegeben werden können. Der operator ist global zu überladen.

Beispiel: [DotA 2, Juliane, 558, player: [Heinrich, 575], [Helmut, 582], [Juliane, 558]]

Hinweis: Um shared_pointer vom `this`-Objekt erzeugen zu können, muss die Klasse `Game` **public** von enable_shared_from_this<Game> erben.

Von der Klasse `Game` werden folgende Klassen abgeleitet.

1.1 RGame

`RGame` ist ein ranked Game und bei `Mode::Ranked` zu erstellen.

RGame(string, shared_ptr<Player>) Setzt Instanzvariablen durch Konstruktor der Basisklasse.

int change(bool won) const Liefert 5 falls `won true` ist, ansonsten -5.

ostream& print(ostream& o) const Gibt das Objekt auf den ostream `o` aus.

Format: Ranked Game: `Game::print(o)`

1.2 UGame

`UGame` ist ein unranked Game und bei `Mode::Unranked` zu erstellen.

UGame(string,shared_ptr<Player>) Setzt Instanzvariablen durch Konstruktor der Basisklasse.

int change(bool) const Liefert immer 0.

ostream& print(ostream& o) const Gibt das Objekt auf den ostream `o` aus.

Format: Unranked Game: `Game::print(o)`

2 Player

Die Klasse `Player` hat folgende **Instanzvariablen**.

string name Name eines Players. (Zur Vereinfachung darf davon ausgegangen werden, dass die Namen der Player-Objekte eindeutig sind.)

int mmr Matchmakingrating eines Players.

shared_ptr<Game> hosted_game Gestartetes Spiel von diesem Spieler.

map<string,weak_ptr<Game>> games Map von Spielen an denen Player teilnimmt.

Hinweis: Um shared_pointer vom `this`-Objekt erzeugen zu können, muss die Klasse `Player` **public** von enable_shared_from_this< *Player* > erben.

Die Klasse `Player` hat folgende **Konstrukturen und Methoden**.

Player(string name, int mmr) Setzt Instanzvariablen, wobei `name` nicht leer sein darf und `mmr` größer gleich 0 und kleiner gleich 9999 sein muss. Sollte ein Parameter nicht den vorgegebenen Werten entsprechen, ist eine Exception vom Typ `runtime_error` zu werfen.

string get_name() const Liefert den Namen des `this`-Objekts.

int get_mmr() const Liefert MMR des `this`-Objekts.

shared_ptr<Game> get_hosted_game() const Liefert `shared_ptr<Game>` auf das gestartete Spiel.

void change_mmr(int n) Addiert `n` zum momemtanen MMR, falls möglich. Sollte `mmr` dabei unter 0 fallen oder über 9999 steigen, wird es auf den jeweiligen Grenzwert gesetzt.

bool host_game(string s, Mode m) Ist `s` leer, ist eine Exception vom Typ `runtime_error` zu werfen. Sollte das `this`-Objekt noch kein Game gestartet haben (`hosted_game` ist `nullptr`), ist (abhänging von Mode `m` (Ranked/Unranked)) ein Spiel vom Typ `RGame` oder `UGame` zu erzeugen, unter `hosted_game` einzutragen und `true` zu liefern. Andernfalls ist `false` zu retournieren.

bool join_game(shared_ptr<Game> g) Falls das `this`-Objekt zum Spiel `g` nicht hinzugefügt werden kann (`Game::add_player` liefert `false`), ist `false` zu retournieren. Sonst ist `g` in `games` (Map der Spiele an denen teilgenommen wird) einzutragen und `true` zu retournieren.

bool leave_game(shared_ptr<Game> g) Entfernt Game `g` aus der Map `games` (Spiele an denen teilgenommen wird) und das `this`-Objekt aus der Map der teilnehmenden Spieler in Game `g`. Liefert `true` falls beides erfolgreich, ansonsten `false`.

vector<weak_ptr<Player>> invite_players(const vector<weak_ptr<Player>>& v) Versucht jeden Player aus `v` zum gestarteten Spiel vom `this`-Objekt einzuladen, also im Game die Player einzuschreiben und bei den `Player`-Objekten Game in den teilnehmenden Spielen einzutragen. Liefert eine Liste aller `weak_ptr` welche entweder ungültig waren oder nicht eingeladen werden konnten.

bool close_game() Falls ein Spiel gestartet ist (`hosted_game` ist nicht `nullptr`), soll dieses freigegeben werden und `true` retourniert werden, ansonsten `false`.

Die Klasse `Player` hat folgendes **Ausgabeformat**.

ostream& print(ostream& o) const Gibt das Objekt auf den ostream `o` aus.

Format: [`name`, `mmr`, hosts: `hosted_game_name`, games: {`Game_name`, `Game_name`, ... }]

operator<< `Player`-Objekte sollen zusätzlich über operator<< ausgegeben werden können. Der operator ist global zu überladen.

Anmerkung: Ist `hosted_game` leer, soll nothing ausgegeben werden.

Beispiel: [Heinrich, 20, hosts: nothing, games{Sims 4, Sims 3, Doom}]

3 GameKey

Die Erklärung warum die Klasse `GameKey` benötigt wird finden Sie auf den Begleitfolien. Die Implementierung von `GameKey` können Sie wie folgt direkt übernehmen.

```
#include "player.h"
class Game;
class GameKey {
    GameKey() {} // Private. Implementierung kann auch in GameKey.cpp erfolgen.
    friend bool Player::join_game(std::shared_ptr<Game>);
    friend bool Player::leave_game(std::shared_ptr<Game>);
};
```

4 Zusatzaufgabe

Die Klasse `GameCard` hat folgende **Methoden und Konstruktor**.

GameCard()

unique_ptr<GameCard> reg() const Erstellt ein `GameCard`-Objekt welches von einem unique_ptr verwaltet wird.

Erweitern Sie Ihr Program um die folgenden Punkte. Anders als bei der Basisimplementierung, sind die weiteren Signaturen der Methoden und genauen Ablauefe nicht exakt vordefiniert und koennen frei gewaehlt werden.

- Erweitern Sie die Klasse Player um eine Instanzvariable unique_ptr<GameCard>.
- Player koennen ihre GameCard an andere Player Objekte weitergeben.
- Ein Ranked Game kann nur mit GameCard betreten werden.
- Beim Aufruf der Methode play für ein Ranked Game sollen beim Bereinigen der Map der teilnehmenden Player zusätzlich jene Player-Objekte entfernt werden, die kein GameCard-Objekt mehr besitzen.