

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 08.12.20**

1 Game

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

string name Name des Spiels.

shared_ptr<Player> host Leiter des Spiels.

map<string, shared_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 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`.

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

bool add_player(const GameKey& gk, shared_ptr<Player> p) Falls bereits ein **Player**-Objekt mit gleichem Namen wie **p** vorhanden ist, liefert die Methode **false**. Ansonsten wird **p** als teilnehmender Spieler hinzugefügt, falls sein MMR sich um weniger als 10% des MMRs von **host** unterscheidet. Im **Player**-Objekt soll in diesem Fall das **Game**-Objekt **nicht** eingetragen werden. Begründung und Erklärung von **GameKey** siehe weiter unten.

shared_ptr<Player> best_player() const Ist **players** leer, so ist eine Exception vom Typ **runtime_error** zu werfen. Ansonsten ist ein Pointer auf das teilnehmende **Player**-Objekt mit höchstem MMR zu liefern. Bei mehreren Objekten, ist das Erste in der Map auftretende zu liefern.

size_t number_of_players() const Liefert Anzahl der teilnehmenden Spieler.

shared_ptr<Player> play(size_t i) Iteriert über alle teilnehmenden Spieler. Der *i*-te Spieler ist der Gewinner. Das MMR aller Verlierer wird um $1 * \text{change}(\text{false})$ geändert. Falls das MMR eines Verlierers, größer als das des Gewinners ist, wird es um $2 * \text{change}(\text{false})$ geändert. **Danach** wird das MMR des Gewinners um $\text{change}(\text{true})$ geändert. Retourniert wird ein Pointer auf den Gewinner.

virtual int change(bool) const = 0 Pure virtual Methode.

virtual ~Game() = default Bei Vererbung 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 zu erzeugen, 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 x) const Liefert 5 falls **x true** ist, ansonsten -5.

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

Format: Ranked Game: **Game->Print**

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`

2 Player

Die Klasse Player hat folgende **Instanzvariablen**.

`string name` Name eines Players.

`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_ptr` vom `this`-Objekt zu erzeugen, 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 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 momentanen MMR, falls möglich. Sollte `mmr` dabei unter 0 fallen oder über 9999 steigen, wird es nicht geändert.

`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 ein Game zu erstellen (abhängig von Mode `m` (Ranked/Unranked)), das Spiel unter `hosted_game` einzutragen und `true` zu liefern. Ansonsten `false`.

`bool join_game(shared_ptr<Game> g)` Falls das `this`-Objekt schon in `Game`-Objekt als Teilnehmer vorhanden, liefert die Methode `false`. Wenn Objekt Game beitreten kann (siehe Teilnahmebedingung in `Game`), wird Objekt bei Game als Teilnehmer eingetragen und in der Map der teilnehmenden Spiele das `Game`-Objekt eingetragen, anschließend wird `true` retourniert.

`bool leave_game(shared_ptr<Game> g)` Entfernt Game `g` aus den teilnehmenden Spielen und das `this`-Objekt aus 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 Spiel gestartet ist, soll es 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

- Sofern Sie den Zusatzpunkt erhalten möchten, beachten Sie bitte, dass in diesem Fall sowohl für die Basisimplementierung, als auch für die Zusatzimplementierung eine automatische **Plagiatsüberprüfung** durchgeführt wird.
- Projektpunkte tragen **nicht** zu den, für einen positiven Abschluss erforderlichen Testpunkten bei, führen aber im Fall einer positiven Bewertung gegebenenfalls zu einer Verbesserung der Note.
- **Abgabe des Zusatzes bis 13.01.21**
- Denken Sie daran sich Ihre Basisimplementierung für den Test zwischen zu speichern bevor Sie mit dem Zusatz beginnen.

Die Klasse `Player` wird um folgende **Instanzvariable** und **Konstruktorparameter** erweitert.

string team Teamname eines Players.

Player(string name, int mmr, string team) Alte Bedingungen gelten weiterhin, zusaetzlich darf `team` nicht leer sein.

Die Klasse `Player` wird um folgende **Methoden** erweitert.

bool operator== Player-Objekte sind gleich, wenn sie ueber den gleichen Namen und Teamnamen verfuegen. Globale Ueberladung erlaubt.

bool operator!= Player-Objekte sind gleich, wenn sie nicht ueber den gleichen Namen und Teamnamen verfuegen. Globale Ueberladung erlaubt.

string get_team() const Liefert Namen des Objekts.

void change_team(string t) Ist `t` leer, ist eine Exception zu werfen vom Typ `std::runtime_error`. Setzt Teamnamen auf `t`.

ostream& print(ostream& o) const Der Teamname soll zusaetzlich ausgegeben werden.

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

Die Klasse `Game` wird um folgende **Instanzvariable** erweitert.

unordered_map<Player,int> hall_of_fame Map von allen Gewinnern dieses Game-Objekts. Am Anfang leer.

shared_ptr<Player> play(size_t i) Eine Kopie des Player-Objekts wird hinzugefuegt als Schluessel. Der Wert zaehlt die Siege des Player-Objekts.

std::pair<string,int> best_team() const Liefert das Team mit den meisten Siegen, sowie die Anzahl der Siege des Teams.

std::pair<Player,int> best_player() const Liefert das Player-Objekt mit den meisten Siegen, sowie die Anzahl der Siege des Player-Objekts.