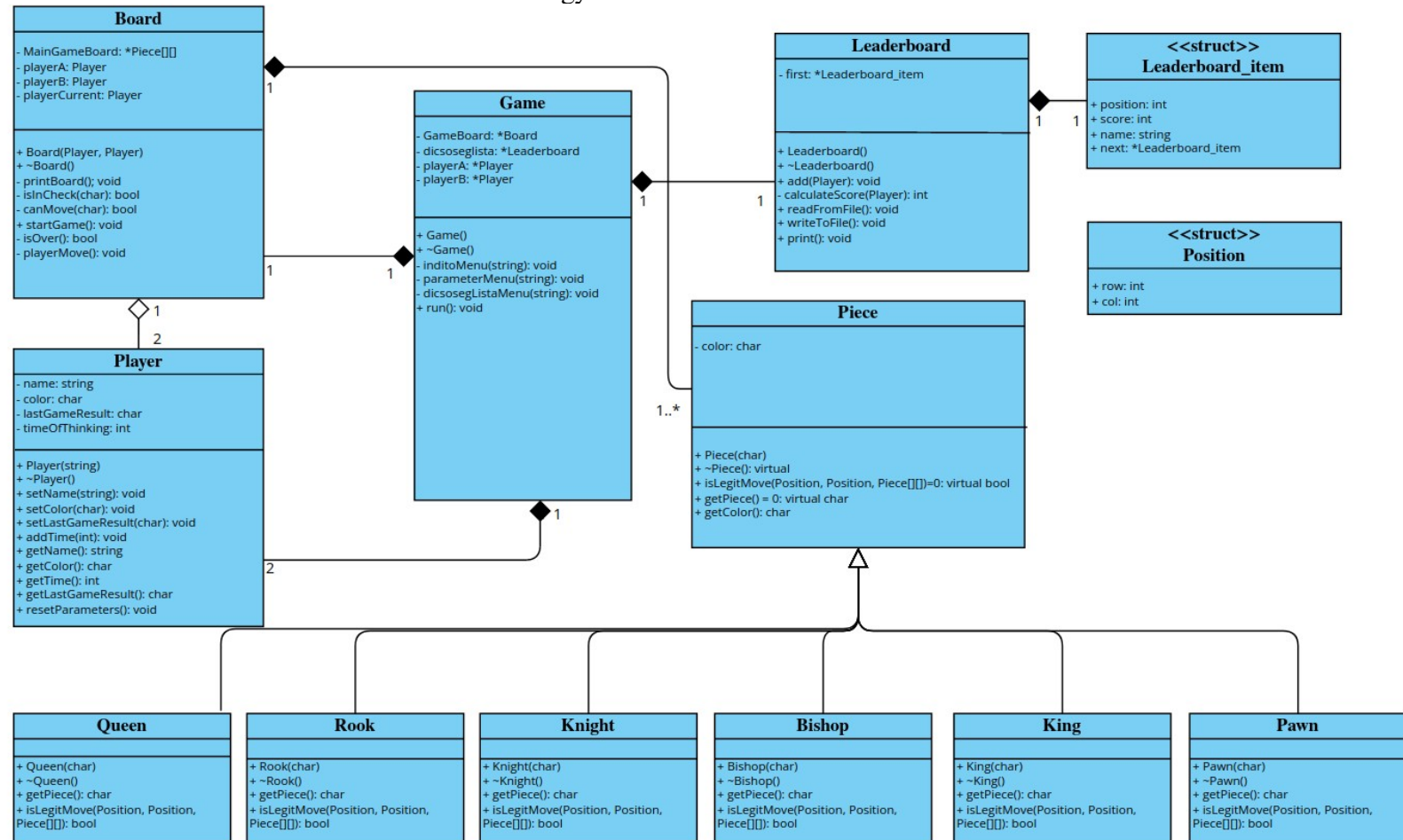


## Sakk játék

A következőkben tárgyalom a sakk játék tervét, melyben a játékban szereplő osztályokat, azok kapcsolatait, illetve az azokban szereplő attribútumokat mutatom be.

A játékot a következő osztálydiagram írja le. Az osztályokat a benne lévő függvényekkel és attribútumokkal az ábra alatti részen tárgyalom.



A játék osztályainak részletesebb leírása:

### Game:

Ez a játék központi eleme. Ezzel lehet a játékot inicializálni, a menü a belső függvények segítségével jelennek meg és így tud a felhasználó interakcióba lépni a programmal.

#### Attribútumai:

**GameBoard: \*Board:** a sakktáblát reprezentálja.

**dicsoseglista: \*Leaderboard:** a játék dicsőséglisátját reprezentálja.

**playerA, playerB: \*Player:** a játék két játékosa.

Függvényei, konstruktora, destruktora:

**Game():** Amikor az osztály e konstruktora meghívódik akkor a játék létrehozza a játékosokat véletlenszerű nevekkal. A játék létrehoz egy Leaderboardot és beolvassa fileből a dicsőséglistát (ha van ilyen fájl).

**~Game():** a játék destruktora: elmenti a a dicsőséglistát a fájlba és felszabadítja a lefoglalt memóriákat: játékosok, dicsőséglista.

A következő menü függvényekkel hívódnak meg a menük, melyekben a program bemenetet kér a felhasználótól. Paraméterük egy string amely a menü kimenetén jelenik meg a hívás után.

**inditoMenu(string): void:** ha a játékos a játék indítását választja, akkor a program itt reseteli a játékosok bizonyos paramétereit és állítja be véletlenszerűen a játékosok színét.

**parameterMenu(string): void:** ebben a menüben állítható be a játékosok nevei.

**dicsoseglistaMenu(string): void:** ez a menü jeleníti meg a dicsőséglistát.

**run(): void:** a Game osztály példányosítása után ezzel a függvénnyel futtatható a játék a menükkal. Ekkor az indítómenü hívódik meg melyben a játékos a többi menüt éri el illetve elindíthatja a játszmat is.

## Piece, és a gyermekosztályai:

A sakkjáték bábújait a Piece osztály reprezentálja. A sakktáblán való elhelyezésükkor a játék ezt a használja statikus típusként. A játéktábla megfelelő helye szerint választ a játék dinamikus típust a gyermekosztályok közül.

### Piece:

Attribútumai:

**color: char:** az adott bábú színe.

Függvényei, konstruktora, destruktora:

**Piece(char):** bábú konstruktora: a paramétere a bábú színe.

**~Piece(): virtual**

**isLegitMove(Position, Position, Piece[][])=0: virtual bool:** egyik tisztán virtuális függvénye mely a gyermekosztályok implementációjában adja vissza, hogy az adott gyermekbábú a első paraméterként kapott pozícióból léphet-e a második paraméterként adott pozícióba azon a kétdimenziós bábúkból álló tömbön ami a sakktáblát reprezentálja.

**getPiece()=0: virtual char:** másik tisztán virtuális függvény. Vissza adja az adott bábút reprezentáló karaktert. Ezek a karakterek a specifikációban tárgyalta módon vannak hozzárendelve.

**getColor(): char:** a bábú színét adja vissza, melyet a konstruktorban kell megadni ('B' vagy 'W').

## Gyermekosztályai:

Egyszerűsítésképp a gyermekosztályokat egy leírásban részletezem, hiszen mindegyik ugyanazzal az attribútummal és ugyanazokkal a függvényekkel rendelkezik csak az implementáció különbözik. A konstruktort és destruktort gyermekek között összevonva *GyermekosztályNév()* és *~GyermekosztályNév()* névvel írom le.

#### Attribútumai:

**color: char:** az adott bábú színe, öröklött attribútum.

#### Függvényei, konstruktora, destruktora:

**GyermekosztályNév(char):** a bábú színét reprezentáló karaktert kell megadni, a szülő Piece(char) konstruktorát hívja meg.

**~GyermekosztályNév()**

**getPiece(): char:** a szülő osztály virtuális függvényét implementálja. Minden bábú azt a karaktert adja vissza ami hozzá van rendelve (specifikációban leírtak szerint).

**isLegitMove(Position, Position, Piece[][]): bool:** ez is a szülő osztály virtuális függvényét implementálja. A különböző bábúknál természetesen különbözik az implementáció, és igazat ad vissza ha az adott bábú az első paraméterként megadott pozícióból a második pozícióba legitim lépést tehet az ugyancsak paraméterként megadott sakktablán.

### **Leaderboard\_item:**

Ez egy struktúra amiből felépül a Leaderboard mint lista.

#### Attribútumai:

**position: int:** jelzi, hogy az adott játékos hanyadik helyen áll a dicsőséglistán.

**score: int:** a játékos pontszámát reprezentálja. Minél kevesebb gondolkodási idő alatt nyert egy játékos valamilyen mértékben annál nagyobb.

**name: string:** a játékos neve.

**next: \*Leaderboard\_item:** a következő dicsőséglista elem. Maximum 10 játékost tárol a Leaderboard, vagyis a 10. játékos után ez mindenképp NULL.

### **Leaderboard:**

A nyertes játékosokat tárolja pontszámaik alapján rendezetten. Ha egy játékos pontja nem éri el az 10. helyen álló játékos pontját akkor nem kerül fel a listára.

#### Attribútumai:

**first: \*Leaderboard\_item:** a dicsőséglista első helyén álló játékost reprezentálja, ezen mentén érhető el a többi játékos a dicsőséglistán.

#### Függvényei, konstruktora, destruktora:

**Leaderboard().**

**~Leaderboard():** végigmegy a lista elemein és felszabadítja a lefoglalt memóriákat.

**calculateScore(Player): int:** az adott játékos pontját számolja ki: minél kevesebb gondolkodási idő alatt nyert az adott játékos annál több pontja lesz.

**add(Player): void:** a listán a megfelelő pozícióba felveszi az adott játékost ügyelve arra, hogy a maximum játékosok száma a dicsőséglistán 10 maradjon.

***readFromFile(): void:*** ha létezik a fájl, akkor beolvassa belőle a mentett dicsőséglistát.

***writeToFile(): void:*** kiírja a fájlba a dicsőséglistát olyan formátumban, hogy megfelelő legyen a beolvasás.

***print(): void:*** a specifikációba már tárgyalt módon jeleníti meg a standard kimeneten a dicsőséglistán szereplő játékosokat és ponjaikat, a helyezésükkel együtt.

## Board:

Ez a játék táblája amin történik a játék menete. Ebben történik a játékoson egymás utáni lépései. A lépések közben a játék számolja a játékosok gondolkodásidejét és addig tart a játék míg az egyik játékos nincs olyan pozícióban, hogy nem tud lépni.

Attribútumai:

***MainGameBoard: \*Piece[[]]:*** a sakktáblát reprezentáló kétdimenziós tömb, melyen bábukra mutató pointerok vannak. A `playerMove()` ezt használja amikor a lépéseket megteszik a játékosok, illetve a `printBoard()` is ennek az állapotát reprezentálja.

***playerA: Player:*** az egyik játékost reprezentálja.

***playerB: Player:*** a másik játékost reprezentálja.

***playerCurrent: Player:*** azt a játékost reprezentálja amelyik éppen a soron következő.

Függvényei, konstruktora, destruktora:

***Board(Player, Player):*** a konstruktor mely létrehozza a táblát és lefoglalja a memóriákat a megfelelő bábuknak.

***~Board():*** felszabadítja a lefoglalt memóriákat.

***printBoard(): void:*** a specifikációban leírt módon jeleníti meg a sakktáblát a standard kimeneten az aktuális játék állással.

***isInCheck(char): bool:*** a függvény paramétere egy karakter mely egy játékos színét reprezentálja. Megkeresi az adott színű játékos királyát és megnézi, hogy van-e olyan másik színű bábú amelynek egy legit lépésével oda lehetne lépni a király helyére (leütni azt).

***canMove(char): bool:*** a függvény paramétere egy karakter mely egy játékos színét reprezentálja. A függvény végig megy az adott színű játékos összes bábuján és megnézi, hogy létezik-e olyan amelynek van olyan legit lépése amelyet ha meglép, akkor nem lesz az adott játékos sakkban. Ha van ilyen igazzal tér vissza egyébként hamis.

***isOver(): bool:*** a függvény minden lépés után ellenőrzi, hogy vége van egy a játéknak. Ha a jelenlegi játékos színére a `canMove(char)` hamis értéket ad vissza, akkor vagy patthelyzet van vagy a játékos mattot kapott, ekkor vége a játéknak (a két eset további eldöntésére ha az adott színű játékosra az `isInCheck(char)` függvény is igazat ad vissza akkor a játékos mattot kapott).

***playerMove(): void:*** ez a függvény kéri be a jelenlegi játékos lépését és ha ez legit lépés akkor megteszi a sakktáblán, ellenkező esetben addig kéri ameddig nem ad legit lépést. Ez a függvény végzi továbbá a játékosok gondolkodási idejének számlálását is melyet a játékosokban rögzít.

***startGame(): void:*** elindítja a játékot: a jelenlegi játékos (kezdetben ez mindig a fehér játékos, az, hogy melyik a fehér véletlenszerűen választott) fog lépni a `playerMove()`-val majd megcseréli a jelenlegi

játékost és ez a játékos is a `playerMove()`-val lép. Minden lépés után a `printBoard()` megjeleníti a megváltozott sakktábla állapotát és addig folytatódnak a lépések ameddig nem igaz az `isOver()`.

## Player:

A játékosokat reprezentáló objektum. Rendelkezik különböző attribútumokkal, melyekkel a játékosok azonosíthatók, illetve a pontjuk számolható: az adott játékos nyert-e, milyen hosszú gondolkodási idővel.

### Attribútumai:

***name: string:*** a játékos nevét reprezentálja. Kezdetben, ha a játékos nem változtatja meg, egy véletlenszerű név.

***color: char:*** a játékos színe. Véletlenszerűen fekete, vagy fehér, ekkor a másik játékos színe az ellenkező.

***lastGameResult: char:*** ez adja meg, hogy a játékos az előző játékban nyert-e vagy nem, illetve döntetlen lett-e. Ezeket a következő karakterek jelölik ebben sorrendben: 'W', 'L', 'D'.

***timeOfThinking: int:*** a játékos a játék során eltöltött gondolkodási idejét tárolja.

### Függvényei, konstruktora, destruktora:

***Player(string):*** paramétere egy string mely a játékos nevét állítja be.

***~Player()***

***setName(string): void:*** a játékos nevét változtatja meg az adott stringre.

***setColor(char): void:*** a játékos színét változtatja meg a megadott karakterre.

***setLastGameResult(char): void:*** a játékos utolsó játékának a kimenetele állítható be. Minden játszma végén meghívja a Board, hogy beállítsa a játszma kimenetelét az egyes játékosoknak.

***addTime(int): void:*** hozzáadja a játékos `timeOfThinking` attribútumában tárolt idejéhez a paraméterként megadott számot.

***getName(): string:*** visszaadja a játékos nevét.

***getColor(): char:*** visszaadja a játékos színét reprezentáló karaktert.

***getTime(): int:*** visszaadja a játékos utolsó játékában akkumulált gondolkodási idejét.

***getLastGameResult(): char:*** visszaadja a játékos utolsó játékának kimenetelét.

***resetParameters(): void:*** ez a függvény visszaállítja a játékos bizonyos attribútumait a következő módon ezekre az értékekre:

color: '0'

lastGameResult: '0'

timeOfThinking: 0