HÁZI FELADAT

Programozás alapjai 2.

Végleges program

Radnai Bálint

C1S8GH

2020. május 18.

TARTALOM

Feladat

Készítsen objektumot a malom nevű játék megvalósításához! Az objektum tárolja a játék állását és "ismerje" a szabályokat, azaz ne engedjen hibás lépést! A figurákat is objektumokkal valósítsa meg!

Demonstrálja a működést külön modulként fordított tesztprogrammal! A játék állását nem kell grafikusan megjeleníteni, elegendő csak karakteresen, a legegyszerűbb formában! A megoldáshoz **ne** használjon STL tárolót!

Feladatspecifikáció

Feladat, játékszabályok

A feladat a malomjáték megvalósítása objektumorientált szemlélettel, C++ nyelven. A játék szabályait a következő weboldal lényegretörően foglalja össze, ez adja a konkrét megvalósítás alapját:
http://mek.niif.hu/00000/00056/html/135.htm[1] (legutóbbi elérés: 2020.05.18.)

Működés

A játék a `malom` nevű program (`malom.exe` futtatható fájl) indításával nyitható meg, és konzolablakban fut. A program a vele egy mappában levő/létrehozott `malom.txt` naplófájlban jegyzi a legutóbbi játék menetét.

Beállítások

A felhasználó a játék elején egy egyszerű szöveges menüből kiválaszthatja, hogy egy- vagy kétjátékos módban kíván-e játszani. Kétjátékos módban egyedül vagy egy társával játszhat, a konzolt felváltva használva. Egyjátékos módban az ellenfél egy – a programba beépített – bot.

A beállításokhoz tartozik az is, hogy szeretne/nem szeretne kezdeni, vagy hogy véletlenszerűen sorsoltatja ki a kezdő játékost. A kezdő játékost és a korongjait világos, a második játékost sötét szín azonosítja.

[1] Egyértelműsítő szabályok: ha valaki a korongletevési fázisban egy lépésben egyszerre két malmot hoz létre, az ellenfélnek akkor is csak legfeljebb egy korongját veheti el.
Malom létrehozásakor nem kötelező az ellenfél valamely korongját elvenni.
A csiki-csuki stratégia is szabályos.
A játék döntetlennel ér véget, ha a játékosok a második fázistól kezdve, egymás utáni tizenhárom lépéspár során nem ütöttek le egyetlen korongot sem.

Lépések

Ezután elindul a játék. A program kezdetben, majd minden lépés után karakteresen megjeleníti az aktuális játékállást, és a játékosok még kézben levő korongjainak számát.

A program a játék fázisa és a játékosok korongjai száma alapján megkérdezi a soron következő játékost, hogy

1. melyik csomópontba szeretne korongot letenni[2], vagy
2. honnan, illetve hova szeretne lépni (csúsztatni), vagy
3. honnan, illetve hova szeretne ugrani.

A játékos a tábla csomópontjaira például az angol ABC betűivel tud hivatkozni.

Ha a játékos a lépésével malmot hozott létre, akkor ezután megadhatja az ellenfél egy korongját, amit le szeretne venni (érvénytelen választás esetén itt is hibaüzenetet küld a program, majd megismételteti a választást). Nem kötelező korongot levenni, a játékos ezt is jelezheti.

Hibás lépés esetén, a program a lépés megtétele helyett a hibára utaló figyelmeztető üzenetet ír ki, és a játékosnak újra be kell írnia a lépése paramétereit.

A játék vége

A játék a szabályok szerint véget ér, ha a soron következő játékos nem tud lépni, vagy ha valamelyik játékosnak csak két korongja marad. Ilyenkor a másik játékos nyer. A lábjegyzetben írt egyértelműsítő szabály értelmében döntetlennel is véget érhet a játszma.

A program kirajzolja a végállapotot, kiírja a játzsma befejezésének okát, és a nyertest.

Tesztelés

A tesztelés egyrészt az elkészült játék konzolos kipróbálását jelenti. Ezenkívül készítek egy teszt fájlt, amit a szabványos bemenetre irányítva, ellenőrizhető a játék megfelelő működése.

Memóriakezelés

A memóriaszivárgás detektálására a *memtrace* ellenőrző programot használtam a megadott módon. A memória tesztje sikeres volt.

[2] (A feladatkiírás értelmében a korongoknak objektumoknak kell lenniük.)

Pontosított feladatspecifikáció

Az eredeti feladatspecifikáció csak apró részletekben módosult.

Pontosítások:

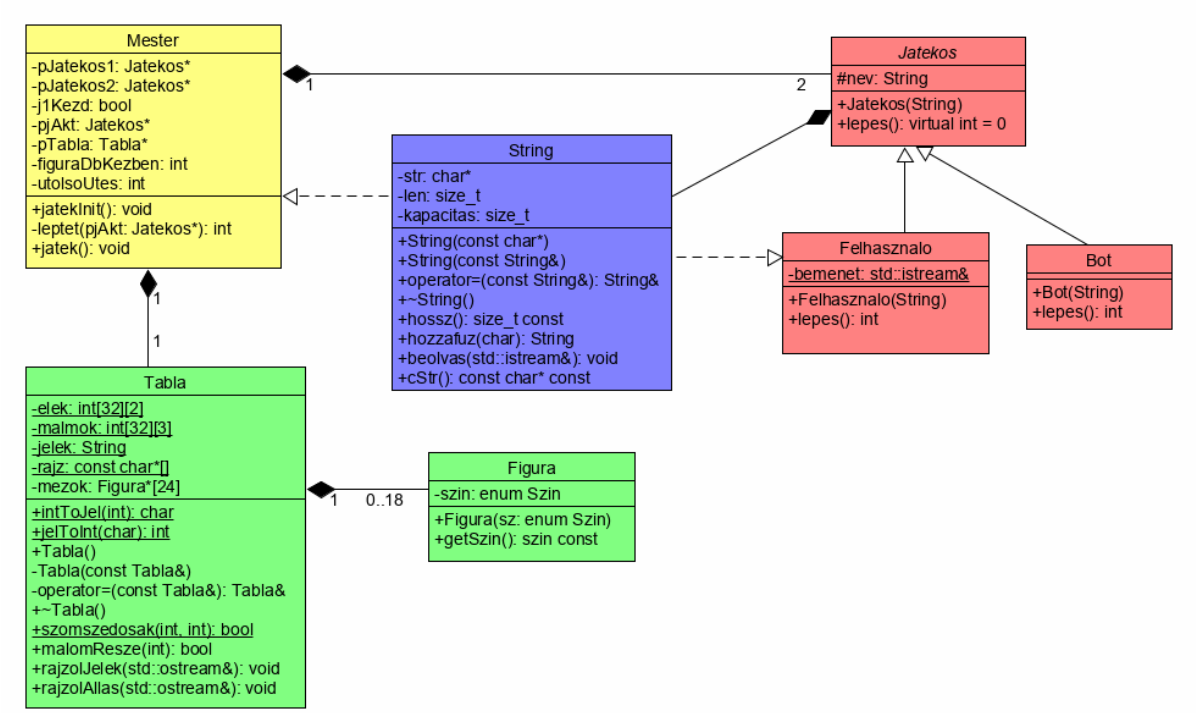
- Ha a játék véget ér, a program új játékot ajánl. Ha a felhasználó elfogadja, akkor teljesen új játék indul, különben a program terminál.
- A játékos a tábla 24 csomópontjára (mezőjére) az angol ABC nagybetűivel tud hivatkozni, A-tól X-ig. Erről a játék elején kap egy szemléletes ábrát/jelmagyarázatot.

Pontosított terv

A feladat a játék osztályhierarchiájának- és a tesztmoduljának megtervezése.

Objektum terv

Ez az ábra mutatja a játékprogram tervezett objektummodelljét.



A modell magára a játékra fókuszál. A játékot egy játékmester (*Mester*) vezényli. A játék beállítási szakaszában dinamikusan létrehozza a két ellenfelet, majd a játéktáblát is. Mindegyikükért ő, a játékmester felel: ha ő megszűnik, megszűnnek a játékosok és a tábla is. A játékot a jatek függvénnyel végzi: kezdetben beállít minden fontos változót (*jatekInit*), majd felváltva lépteti a játékosokat (malom esetén egy extra ág a koronglevétel). Ellenőrzi a lépéseket, és leállítja a játékot, ha a végfeltételek teljesülnek.

A játékosok (*Jatekos*) normál felhasználók (*Felhasznalo*) vagy beépített robotok (*Bot*) lehetnek. Szerepük ugyanaz, de a lépésmód más. Emiatt célszerű volt öröklést használni, és a *Jatekos* osztály absztrakt ősosztállyá tenni, a lépés (*lepes*) függvényt pedig tisztán virtuálisként megvalósítani. A felhasználó a standard bemeneten kommunikál a programmal, ezért az input stream-et statikus tag jegyzi.

A játéktábla nem másolható/értékadható, mivel a modell szempontjából felesleges lenne. A tábla 24 mezejét a program a 0…23 számokkal azonosítja. A felhasználó számára az angol ABC nagybetűi jelennek meg, amelyeket – sorrendben – a *jelek String* tartalmazza. A konverziót függvények oldják meg.

A tábla elrendezése állandó. A táblát, mint gráfot, vagyis a gráf éleit (*elek*) éllistaként, a legegyszerűbb módon, rendezetten tárolom. A 16 malom/malomhely gyorsabb eléréséhez redundanciát alkalmazok: minden egyes csúcshoz a 0.-tól 23.-ig tárolom azt a két malmot, amelyikben az adott kódú csúcs is szerepel. A malmokat is rendezve tárolom, így a *poz*. csomópont pontosan a *2*poz*. (vízszintes) és a *2*poz+1*. (függőleges) malmokban szerepel. Ezt az információt statikus tagokban tárolom. Erre az adatszerkezetre építve, a *szomszedosak* függvény határozza meg két csomópontról, hogy szomszédosak-e, a *malomResze* függvény adja meg azt, hogy egy megadott mező malomban van-e, a *beragadt* függvény pedig azt, hogy a soron következő játékosnak van-e szabályos tolása a táblán az aktuális játékállás alapján. (Ezekre a függvényekre van a legnagyobb szükség a játék 2-3. fázisában és a végállapot ellenőrzésekor.)

A mindenkori állást a *mezok* tömb tartja nyilván. A tábla default konstruktora alapértelmezetten NULL pointerekkel tölti fel őket. Új figura letevésekor dinamikus memóriát foglal. *Figura* mozgatásakor csak pointerérték-cserék szükségesek. A figurák egyszerű objektumok: a színüket globális enum típusként (*Szin*) tárolják, de külön kirajzoló tagfüggvényük van.

Végül, mivel több osztály is tartalmaz/hivatkozik karaktertömböt, szükséges egy String típus is. Elég, ha a bementről sort be tudunk beolvasni vele, illetve a legalapvetőbb funkciók meglegyenek benne (STL tároló nem használható).

Algoritmusok

A játék algoritmusai általában

A játékhoz nincs szükség bonyolult algoritmusokra: leginkább esetszétválasztások kellenek.

A játéktábla gráfjában az éleket bináris kereséssel keresem meg. Ez csak minimálisan okoz több lépést, mintha csúcslistával tárolnám őket. Az adatszerkezet sokkal egyszerűbb marad, és a programkódban bőven elfér a játéktábla, nem szükséges fájlból olvasni.

A lépések esetszétválasztását segédfüggvényekbe szervezem. A normál felhasználók lépései bekérésekor egy-egy karakter beolvasása is elegendő, de szabályellenes lépés esetén a játékmester újra lépteti őket.

A botok algoritmusai

Ebben a megvalósításban a botok nem jegyzik meg a pozíciót, amit már lépésre kiválasztottak, hanem paraméterként megkapják a játékmestertől (ha nem _HOVA típusú a lépés, akkor a mester -1-et küld, és a játékos sem veszi figyelembe). Direkt olyan mezőt választanak tolás, illetve ugrás kezdőpozíciójául, ahonnan tudnak igazán jó helyre lépni. A következő lépésben nézik meg azt, hogy azok közül melyik a legjobb. Így nem esnek el a legjobb lehetőségektől.

A **robot** a lépései során **prioritással lát el minden szóba jövő pozíciót**, és (maximumkereséssel) véletlenszerűen választ egyet a legjobb prioritásúak közül.

Tehát először létrehoz egy 24 elemű tömböt, amelynek minden elemét 0-val inicializálja. A 0 prioritás annak felel meg, hogy oda nem léphet abban a lépésben a bot. Ezután végigmegy a mezőkön egy ciklussal, és ha az adott mezőre a szabályok alapján léphet, akkor kiszámolja a mező prioritását. A prioritás ebben az esetben 1-től 5-6-7-ig terjedő pozitív egész, minél nagyobb értéke jelenti a minél kedvezőbb pozíciót a robot szemszögéből.

Lépéstípusonként eltérő, hogy milyen esetszétválasztás alapján számol, de általánosan igaz, hogy ha valamelyik mezőre téve ott malmot alakíthat ki, akkor azt maximális prioritással látja el, illetve második legnagyobb prioritással azokat, ahová téve (vagy lépve/ugorva, vagy pedig ahonnan elvéve) az ellenfél malomkísérletét meghiúsíthatja. Ezek után még célszerűnek tart oda tenni, ahol az ellenfélnek van legalább két korongja, miközben saját magának nincs (vagy legfeljebb egy van). Különben lényegében mindegy, hogy hova tesz. Néha üres helyeket is szívesen elfoglal, máskor pedig igyekszik az ellenfél korongjaihoz közel maradni.

PROGRAMOZÓI DOKUMENTÁCIÓ

Tartalomjegyzék

Osztályok dokumentációja

Osztálylista

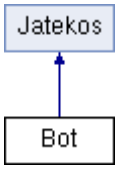
Az összes osztály és struktúra listája rövid leírásokkal:

Bot osztályreferencia

Robotot/számítógépet megvalósító Jatekos-leszármazott. Algoritmikusan számítja a lépések helyét.

`#include <jatekos.h>`

A Bot osztály származási diagramja:



Publikus tagfüggvények

- **Bot** (const **String** &bNev)
- int **lepes** (std::ostream &os, const **Tabla** &tabla, const **Szin** &aktSzin, const **LepesTipus** &tipus, int honnan) const

---

Részletes leírás

Robotot/számítógépet megvalósító Jatekos-leszármazott.
Algoritmikusan számítja a lépések helyét.

Konstruktorok és destruktorok dokumentációja

Bot::Bot (const String & *bNev*) `[inline]`

Konstruktor, beállítja a bot nevét.

Tagfüggvények dokumentációja

int Bot::lepes (std::ostream & *os*, const Tabla & *tabla*, const Szin & *aktSzin*, const LepesTipus & *tipus*, int *honnan*) const `[virtual]`

A bot lépését számító függvény. Az egyes lépéstípusoknál különbözõ feltételekkel, de alapvetõen hasonlóan számol. A lehetséges pozícióknak prioritásértéket ad, majd a maximális prioritású opciók közül véletlenszerûen választ. Odafigyel a malmok kihasználására, az ellenfél malmainak megakadályozására, illetve a lehetõségei csökkentésére; ennél több stratégiája nincsen.

Paraméterek

os	output adatfolyam, ahol a játékmester ír a felhasználóknak
tabla	konstans referencia a táblára, amin lépni kell
aktSzin	az aktuális körben lépõ játékos színe
tipus	a lépés típusa
honnan	az elõzõ lépésben megjelölt pozíció (honnan-hova párok esetén használjuk)

Visszatérési érték

A mezõ kódja, amit a játékos a lépésében megjelölt (ELVESZ esetén -1 is lehet).

Megvalósítja a következõket: **Jatekos** (*o.13*).

Felhasznalo osztályreferencia

Felhasználó, azaz élõ ember játékos, a **Jatekos** osztály leszármazottja. Feladata a játékos nevét tárolni, és konzolon bekérni a lépéseit.

`#include <jatekos.h>`

A Felhasznalo osztály származási diagramja:

Publikus tagfüggvények

- **Felhasznalo** (const **String** &fNev, std::istream &inputStream)
- int **lepes** (std::ostream &os, const **Tabla** &tabla, const **Szin** &aktSzin, const **LepesTipus** &tipus, int honnan) const

---

Részletes leírás

Felhasználó, azaz élõ ember játékos, a **Jatekos** osztály leszármazottja. Feladata a játékos nevét tárolni, és konzolon bekérni a lépéseit.

Konstruktorok és destruktorok dokumentációja

Felhasznalo::Felhasznalo (const String & *fNev*, std::istream & *inputStream*) `[inline]`

Konstruktor, beállítja az input adatfolyamot, és a felhasználó nevét.

Tagfüggvények dokumentációja

int Felhasznalo::lepes (std::ostream & *os*, const Tabla & *tabla*, const Szin & *aktSzin*, const LepesTipus & *tipus*, int *honnan*) const `[virtual]`

A felhasználó lépését bekérõ függvény. Konzolablakból várja a karakterkódot, amit ellenõriz, és int-té alakít.

Paraméterek

os	output adatfolyam, ahol a játékmester ír a felhasználóknak
tabla	konstans referencia a táblára, amin lépni kell
aktSzin	az aktuális körben lépõ játékos színe
tipus	a lépés típusa
honnan	az elõzõ lépésben megjelölt pozíció (honnan-hova párok esetén használjuk)

Visszatérési érték

A mezõ kódja, amit a játékos a lépésében megjelölt (ELVESZ esetén -1 is lehet).

Megvalósítja a következõket: **Jatekos** (*o.13*).

Figura osztályreferencia

A játék figuráit/korongjait megvalósító osztály.
`#include <tabla.h>`

Publikus tagfüggvények

- **Figura** (const **Szin** &fSzin)
- **Szin getSzin** () const
  Getter a figura színéhez.

- void **rajzol** (std::ostream &os)

Részletes leírás

A játék figuráit/korongjait megvalósító osztály.

Konstruktorok és destruktorok dokumentációja

Figura::Figura (const Szin & *fSzin*) `[inline]`

A figura konstruktora, beállítja a színét.

Paraméterek

fSzin	a figura leendõ színe

Tagfüggvények dokumentációja

Szin Figura::getSzin () const `[inline]`

Getter a figura színéhez.

Visszatérési érték

a figura színe

void Figura::rajzol (std::ostream & *os*)

Egy 'V'/'S' karakterrel jeleníti meg a figurát.

Paraméterek

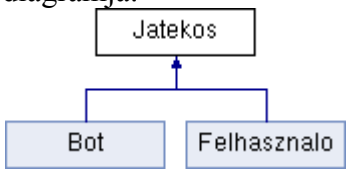
os	az output stream, ahova kiírja a karaktert

Jatekos osztályreferencia

Absztrakt õsosztály a malom játékosaihoz.

`#include <jatekos.h>`

A Jatekos osztály származási diagramja:



Publikus tagfüggvények

- **Jatekos** (const **String** &jNev)
- const **String** &**getNev** () const
- virtual int **lepes** (std::ostream &os, const **Tabla** &tabla, const **Szin** &aktSzin, const **LepesTipus** &tipus, int honnan) const =0
- virtual **~Jatekos** ()

Védett attribútumok

- **String nev**

Részletes leírás

Absztrakt õsosztály a malom játékosaihoz.

Konstruktorok és destruktorok dokumentációja

Jatekos::Jatekos (const String & *jNev*) `[inline]`

Az õsosztály konstruktora

Paraméterek

jNev	**String**, a játékos leendõ nevét tartalmazza

virtual Jatekos::~Jatekos () `[inline]`, `[virtual]`

Virtuális destruktor

Tagfüggvények dokumentációja

String Jatekos::getNev () const `[inline]`

Getter a játékos nevéhez

Visszatérési érték

a játékos neve, **String**

virtual int Jatekos::lepes (std::ostream & *os*, const Tabla & *tabla*, const Szin & *aktSzin*, const LepesTipus & *tipus*, int *honnan*) const `[pure virtual]`

Virtuális függvény, ami a játékosok lépés függvényeit határozza meg.

Paraméterek

os	output adatfolyam, ahol a játékmester ír a felhasználóknak
tabla	konstans referencia a táblára, amin lépni kell
aktSzin	az aktuális körben lépõ játékos színe
tipus	a lépés típusa
honnan	az elõzõ lépésben megjelölt pozíció (honnan-hova párok esetén használjuk)

Visszatérési érték

A mezõ kódja, amit a játékos a lépésében megjelölt (ELVESZ esetén -1 is lehet).

Megvalósítják a következõk: **Bot** (*o.10*) és **Felhasznalo** (*o.11*).

Adattagok dokumentációja

String Jatekos::nev `[protected]`

a játékos neve, tartalmazhat szóközöket is

Mester osztályreferencia

Játékmester osztály.
`#include <mester.h>`

Publikus tagfüggvények

- **Mester** ()
- **~Mester** ()
- void **jatek** ()

Részletes leírás

Játékmester osztály.

Felelõs a játékosok és a tábla kezeléséért (dinamikusan foglalja õket, megszûnésekor õket is megszünteti), és a malom szabályai szerinti játék lebonyolításáért.

Konstruktorok és destruktorok dokumentációja

Mester::Mester () `[inline]`

A játékmester default konstruktora. Így kiírva, explicit is hívható (**Mester()**).

Mester::~Mester ()

Destruktor. A - dinamiusan foglalt - játékosokat és táblát szabadítja fel.

Tagfüggvények dokumentációja

void Mester::jatek ()

A játékmester legfontosabb függvénye, magát a játékot valósítja meg.

String osztályreferencia

```
#include <mystring.h>
```

Publikus tagfüggvények

- **String** ()
- **String** (const char ch)
- **String** (const char *str)
- **String** (const **String** &theOther)
- **~String** ()
- **String** & **operator=** (const **String** &rhs)

- size_t **size** () const
- size_t **capacity** () const
- char & **operator[]** (size_t idx)
- const char & **operator[]** (size_t idx) const
- **String** & **append** (const char ch)
- const char * **c_str** () const

Részletes leírás

String osztály nullterminált dinamikus karaktertömbök tárolására.
Szabad kapacitással dolgozik, okosabban növelve a méretét.

Konstruktorok és destruktorok dokumentációja

String::String ()

Paraméter nélküli konstruktor:

String::String (const char *ch*)

Konstruktor egy karakterbõl

Paraméterek

ch	- karakter, amibõl az új String-et létrehozza

String::String (const char * *str*)

Konstruktor nullterminált karaktertömbbõl

Paraméterek

ch	- nullterminált karaktertömb, amibõl az új String-et létrehozza

String::String (const String & *theOther*)

Másoló konstruktor

Paraméterek

konstans	referencia a másolandó String-re

String::~String () `[inline]`

Destruktor.

Tagfüggvények dokumentációja

String & String::append (const char *ch*)

String-hez jobbról karaktert fûz

Paraméterek

rhs	- a meglévõ String-hez jobbról hozzáfûzendõ karakter

Visszatérési érték

az új, összefûzött **String**

const char* String::c_str () const `[inline]`

C-sztringet ad vissza

Visszatérési érték

pointer a tárolt, vagy azzal azonos tartalmú nullával lezárt sztring-re.

size_t String::capacity () const `[inline]`

Sztring kapacitását adja vissza.

Visszatérési érték

sztring maximális kapacitása (lezáró nulla nélkül).

String & String::operator= (const String & *rhs*)

operator=

Paraméterek

konstans	referencia a jobb oldali String-re

Visszatérési érték

referencia az új értéket kapott String-re

char & String::operator[] (size_t *idx*)

Indexoperátor Túlindexelés esetén std::out_of_range kivételt dob.

Paraméterek

idx	- a nemnegatív index

Visszatérési érték

referencia az index által elért karakterre

const char & String::operator[] (size_t *idx*) const

Konstans indexoperátor Túlindexelés esetén std::out_of_range kivételt dob.

Paraméterek

idx	- a nemnegatív index

Visszatérési érték

referencia az index által elért karakterre

size_t String::size () const `[inline]`

Sztring hosszát adja vissza.

Visszatérési érték

sztring tényleges hossza (lezáró nulla nélkül).

Tabla osztályreferencia

A malom játéktábláját megvalósító osztály.
`#include <tabla.h>`

Publikus tagfüggvények

- **Tabla** (std::ostream &outputStream)
- **Tabla** (const **Tabla** &eredeti)
- **Tabla** & **operator=** (const **Tabla** &eredeti)
- **~Tabla** ()
- **Figura** *& **operator[]** (int poz)
- const **Figura** * **operator[]** (int poz) const
- bool **malomResze** (int poz) const
- bool **mindMalombanVan** (const **Szin** &aktSzin) const
- bool **vanUresSzomszedja** (int poz) const
- bool **beszorult** (const **Szin** &aktSzin) const

Statikus publikus tagfüggvények

- static char **pozToJel** (int i)
- static int **jelToPoz** (char ch)
- static const **String** & **getJelek** ()
- static int **malmokIndex** (int malomIdx, int mezoIdx)
- static bool **szomszedosak** (int poz1, int poz2)
- static void **rajzolJelek** (std::ostream &os)
- static void **rajzolAllas** (const **Tabla** &tabla)

Részletes leírás

A malom játéktábláját megvalósító osztály.

Konstruktorok és destruktorok dokumentációja

Tabla::Tabla (std::ostream & *outputStream*)

A tábla konstruktora. Beállítja a rajzolás helyét (outputStream), és NULL-okkal tölti fel a mezõk tömbjét.

Tabla::Tabla (const Tabla & *eredeti*)

Másoló konstruktor

Paraméterek

eredeti	a másik tábla, amelyrõl a másolat készül

Tabla::~Tabla ()

Destruktor, felszabadítja a tábla nem NULL mezõit.

Tagfüggvények dokumentációja

bool Tabla::beszorult (const Szin & *aktSzin*) const

Megvizsgálja, hogy a megadott színû játékos tud-e szabályos tolást végezni (szomszédos mezõre) valamely korongjával.

Paraméterek

aktSzin	a játékos színe

Visszatérési érték

Pontosan akkor igaz, ha nincs figura, amelyikkel tudna szomszédos mezõre lépni. Ha nincs adott színû figura, akkor is false-szal tér vissza.

const String & Tabla::getJelek () `[static]`

Megadja a mezõkódok sztringjét.

Visszatérési érték

konstans referencia a kódok sztringjére

int Tabla::jelToPoz (char *ch*) `[static]`

Egy mezõkódot indexre konvertál, a '-' jelhez pedig a -1-et rendeli. Túlindexelés esetén std::out_of_range kivételt dob.

Paraméterek

ch	a karakteres mezõkód (vagy '-')

Visszatérési érték

a mezõ indexe (0-tól 23-ig terjed), vagy -1

int Tabla::malmokIndex (int *malomIdx*, int *mezoIdx*) `[static]`

Az osztályszintû malmok tömböt konstans módon indexelõ függvény.

Paraméterek

malomIdx	a malom sorszáma
mezoIdx	a mezõ sorszáma a malmon belül

Visszatérési érték

a malomIdx-edik malom mezoIdx-edik mezõje

bool Tabla::malomResze (int *poz*) const

Megmadja, hogy az adott pozíciójú mezõn szerepel-e olyan figura, amely éppen malomban van.

Paraméterek

poz	a mezõ indexe

Visszatérési érték

Igaz/hamis attól függõen, hogy malom része-e a mezõ.

bool Tabla::mindMalombanVan (const Szin & *aktSzin*) const

Megvizsgálja, hogy a játékos korongjai között van-e olyan, amelyik épp nincs malomban.

Paraméterek

aktSzin	a játékos figurái színe

Visszatérési érték

Pontosan akkor igaz, ha mindegyik figura valamely malom részét alkotja.

Tabla & Tabla::operator= (const Tabla & *eredeti*)

Értékadó operátor

Paraméterek

eredeti	a másik tábla, amelyrõl a másolat készül

Visszatérési érték

Referencia a felülírt táblára, hogy láncolható legyen az értékadás.

Figura *& Tabla::operator[] (int *poz*)

A tábla index operátorai, a mezõket teszik elérhetõvé.

const Figura * Tabla::operator[] (int *poz*) const

char Tabla::pozToJel (int *i*) [static]

Egy mezõindexhez megadja a karakterkódját, a -1-hez pedig a '-' kötõjelet rendeli. Túlindexelés esetén std::out_of_range kivételt dob.

Paraméterek

i	az index (vagy -1)

Visszatérési érték

a karakter (A-tól X-ig terjed), vagy kötõjel

void Tabla::rajzolAllas (const Tabla & *tabla*) [static]

Kirajzolja a táblát a "kimenet"-re.

Paraméterek

tabla	konstans referencia a kirajzolandó táblára

void Tabla::rajzolJelek (std::ostream & *os*) [static]

Jelmagyarázat; kirajolja a játéktáblát a mezõk kódjaival a <kimenet> adatfolyamra.

Paraméterek

os	szabványos output stream, ahova rajzol

bool Tabla::szomszedosak (int *poz1*, int *poz2*) [static]

Megvizsgálja, hogy két, pozíciójukkal adott mezõ szomszédos-e a játáktáblán. Túlindexeléskor std::out_of_range kivételt dob.

Paraméterek

poz1	az egyik mezõ indexe
poz2	a másik mezõ indexe

Visszatérési érték

Szomszédos mezõk esetén true, különben false

bool Tabla::vanUresSzomszedja (int *poz*) const

Megvizsgálja, hogy az adott figura (ha van) tud-e szomszédos üres mezõre lépni.

Paraméterek

poz	a pozíció/mezõindex

Visszatérési érték

Pontosan akkor igaz, ha van ott figura, és tud szomszédos mezõre lépni.

Fájlok dokumentációja

Fájllista

Az összes fájl listája rövid leírásokkal:

jatekos.cpp fájlreferencia

```
#include <cstdlib>
#include <cctype>
#include <cstring>
#include <ctime>
#include "memtrace.h"
#include "jatekos.h"
```

Enumerációk

- enum **Mezo** { **U**, **V**, **S** }

Részletes leírás

A játékosok nem inline függvényeinek definíciói

Enumerációk dokumentációja

enum Mezo

Enumeráció-értékek:

U	Üres mezőt reprezentál.
V	Világos figura van a mezőn.
S	Sötét figura van a mezőn.

jatekos.h fájlreferencia

```
#include <iostream>
#include "tabla.h"
#include "mystring.h"
```

Osztályok

- class **Jatekos**
  Absztrakt õsosztály a malom játékosaihoz.
- class **Felhasznalo**
  *Felhasználó, azaz élõ ember játékos, a **Jatekos** osztály leszármazottja. Feladata a játékos nevét tárolni, és konzolon bekérni a lépéseit.*
- class **Bot**
  Robotot/számítógépet megvalósító Jatekos-leszármazott. Algoritmikusan számítja a lépések helyét.

Enumerációk

- enum **LepesTipus** { **LETESZ**, **TOLAS_HONNAN**, **TOLAS_HOVA**, **UGRAS_HONNAN**, **UGRAS_HOVA**, **ELVESZ** }
  Felsorolt típus az egyes lépések elkülönítésére. A játék elsõ fázisában LETESZ-, a másodikban TOLAS_HONNAN- és TOLAS_HOVA-, a harmadikban pedig UGRAS_HONNAN- és UGRAS_HOVA lepesek következhetnek. Bármelyik lépés után ELVESZ lépés is jön, ha malom keletkezett.

Részletes leírás

A malom játékostípusainak (játékos <- felhasználó / bot) deklarációi

---

Enumerációk dokumentációja

enum LepesTipus

Felsorolt típus az egyes lépések elkülönítésére. A játék elsõ fázisában LETESZ-, a másodikban TOLAS_HONNAN- és TOLAS_HOVA-, a harmadikban pedig UGRAS_HONNAN- és UGRAS_HOVA lepesek következhetnek. Bármelyik lépés után ELVESZ lépés is jön, ha malom keletkezett.

Enumeráció-értékek:

LETESZ	Figura letevése a játéktáblára.
TOLAS_HONNAN	Szomszédos mezőre tolni kívánt figura kiválasztása.
TOLAS_HOVA	Figura tolása szomszédos mezőre: a célmező választása.
UGRAS_HONNAN	Ugró figura kiválasztása.
UGRAS_HOVA	Ugrás figurával egy tetszőleges üres mezőre: a célmező választása.
ELVESZ	Az ellenfél egy figurájának opcionális le-/elvétele a tábláról.

malom_teszt.cpp fájlreferencia

```
#include <iostream>
#include <stdexcept>
#include "memtrace.h"
#include "mester.h"
```

Függvények

- void **tesztJatek** ()
- int **main** ()

Részletes leírás

Egyszerû tesztmodul a malom játékhoz. Ki lehet próbálni benne magát a játékot. Próbajátékban is tesztelhetõ a mûködés.

Függvények dokumentációja

int main ()

Elindítja a programot. I/N karakterrel választhatunk, hogy magát a játékot szeretnénk indítani, vagy tesztelni szeretnénk a tesztadatokkal.

void tesztJatek ()

mester.h fájlreferencia

```
#include <iostream>
#include "jatekos.h"
#include "tabla.h"
```

Osztályok

- class **Mester**
  Játékmester osztály.

Enumerációk

- enum **Allapot** { **JATEK**, **BESZORULT**, **ELFOGYOTT**, **DONTETLEN** }
  A játékmenetek állapotai. Csak a JATEK állapotban folytatódik a játék.

Részletes leírás

A malom játékmesterének deklarációja

Enumerációk dokumentációja

enum Allapot

Az egyes játszmák állapotai. Csak a JATEK állapotban folytatódik a játék.

Enumeráció-értékek:

JATEK	A játék folyamatban van, a soron következő játékos tud szabályosan lépni.
BESZORULT	Az éppen következő játékos beszorult: nem tud szabályosan tolni.
ELFOGYOTT	Az éppen következő játékosnak már cask két figurája maradt a táblán.
DONTETLEN	A letevési fázist követően 13 lépéspár telt el ütés nélkül: a játszma döntetlen.

mystring.cpp fájlreferencia

```
#include <iostream>
#include <iomanip>
#include <stdexcept>
#include <cstring>
#include <cctype>
#include "memtrace.h"
#include "mystring.h"
```

Függvények

- std::ostream & **operator**<< (std::ostream &os, const **String** &rhs)
- std::istream & **operator**>> (std::istream &is, **String** &rhs)
- void **getline** (std::istream &is, **String** &str)

Részletes leírás

A **String** osztály nem inline tagfüggvényeinek a definícióit tartalmazza.

Függvények dokumentációja

void getline (std::istream & *is*, String & *str*)

Teljes sort String-be beolvasó függvény.

Paraméterek

is	- std::istream típusú objektum, ahonnan a String-et olvassa
rhs	- referencia a String-re, ahova olvassuk a sort

std::ostream& operator<< (std::ostream & *os*, const String & *rhs*)

Inserter operátor String-ek kiírásához

Paraméterek

os	- std::ostream típusú objektum, ahová a String-et írja
rhs	- konstans referencia a kiírandó String-re

Visszatérési érték

referencia az std::ostream objektumra a láncolhatóságért

std::istream& operator>> (std::istream & *is*, String & *rhs*)

Extractor operátor String-ek beolvasásához

Paraméterek

is	- std::istream típusú objektum, ahonnan a String-et olvassa
rhs	- referencia a beolvasandó String-re

Visszatérési érték

referencia az std::istream objektumra a láncolhatóságért

mystring.h fájlreferencia

`#include <iostream>`

Osztályok

- class **String**

Függvények

- std::ostream & **operator**<< (std::ostream &os, const **String** &rhs)
- std::istream & **operator**>> (std::istream &is, **String** &rhs)
- void **getline** (std::istream &is, **String** &str)

Részletes leírás

A **String** osztály interfészét tartalmazza. Az 5. heti laboron elkészített osztály átalakított változata.

Függvények dokumentációja

void getline (std::istream & *is*, String & *str*)

Teljes sort String-be beolvasó függvény.

Paraméterek

is	- std::istream típusú objektum, ahonnan a String-et olvassa
rhs	- referencia a String-re, ahova olvassuk a sort

std::ostream& operator<< (std::ostream & *os*, const String & *rhs*)

Inserter operátor String-ek kiírásához

Paraméterek

os	- std::ostream típusú objektum, ahová a String-et írja
rhs	- konstans referencia a kiírandó String-re

Visszatérési érték

referencia az std::ostream objektumra a láncolhatóságért

std::istream& operator>> (std::istream & *is*, String & *rhs*)

Extractor operátor String-ek beolvasásához

Paraméterek

is	- std::istream típusú objektum, ahonnan a String-et olvassa
rhs	- referencia a beolvasandó String-re

Visszatérési érték

referencia az std::istream objektumra a láncolhatóságért

tabla.h fájlreferencia

```
#include <iostream>
#include "mystring.h"
```

Osztályok

- class **Figura**
  A játék figuráit/korongjait megvalósító osztály.
- class **Tabla**
  A malom játéktábláját megvalósító osztály.

Enumerációk

- enum **Szin** { **VILAGOS**, **SOTET** }
  Felsorolt típus a figurák (és a játékosok) színeinek.

Függvények

- **Szin szinEllentett** (const **Szin** &eredeti)
- **String szovegesSzin** (const **Szin** &szin)
- bool **operator==** (const **Figura** &fig, const **Szin** &sz)
- bool **operator!=** (const **Figura** &fig, const **Szin** &sz)

Részletes leírás

A játéktábla deklarációja és a figura típus megvalósítása

Enumerációk dokumentációja

enum Szin

Felsorolt típus a figurák (és a játékosok) színeinek.

Enumeráció-értékek:

VILAGOS	Világos színű figurát vagy játékost jelöl.
SOTET	Sötét figura vagy játékos.

Függvények dokumentációja

bool operator!= (const Figura & *fig*, const Szin & *sz*) [inline]

Egyenlõség operátor, ami egy figura színének különbözõségét vizsgálja egy adott színnel.

Paraméterek

fig	a figura
sz	a szín, amihez hasonlítjuk a figura színét

Visszatérési érték

Egyezés esetén false, különbözõség esetén true.

bool operator== (const Figura & *fig*, const Szin & *sz*) [inline]

Egyenlõség operátor, ami egy figura színének egyezõségét vizsgálja egy adott színnel.

Paraméterek

fig	a figura
sz	a szín, amihez hasonlítjuk a figura színét

Visszatérési érték

Egyezés esetén true, különben false

Szin szinEllentett (const Szin & *eredeti*) `[inline]`

Visszatér egy enum Szin ellentettjével (VILAGOS / SOTET).

Paraméterek

eredeti	a megadott szín

Visszatérési érték

az ellentétes szín

String szovegesSzin (const Szin & *szin*) `[inline]`

A szín szöveges nevét adja vissza ("Világos" / "Sötét").

Paraméterek

eredeti	a szín

Visszatérési érték

a szín szövegesen