Házi feladat

Programozás alapjai 2.

Végleges

Tömöri Péter András
I4RZ0O

2023. május 28.

Tartalom

1. Feladatspecifikáció

A program neve Mancala, célja pedig, hogy az egyező nevű táblajátékot reprezentálja konzolos felületen, az objektumorientált programozás lehetőségeit használva.

A játékot a felhasználó több fajta játékossal is képes lesz játszani: akár egy *másik felhasználóval*, vagy egy előre meghatározott nehézségű, okos algoritmust használó *számítógép ellen*. Továbbá a kettő kombinációjára is lehetőség van: a *co-pilot módban* a felhasználó hozza meg a végső döntést a lépéshez, de van mellette egy okos számítógép előre megadott szinttel, aki javaslatot ad, hogy szerinte melyik cellát lenne érdemes választani.

1.1 Játékszabályok

- A tábla cellákból fog állni, mindkét játékos oldalán ugyanakkora mennyiségű (1-15, hivatalosan 6db), ezek egymással szemben vannak.
- A két sor végén 1-1 bázissal, a játékostól jobbra levő bázis a sajátja.
- A játék kezdetekor minden cellában azonos mennyiségű (1-99, hivatalosan 4 db) üveggolyó van.
- A soron következő játékos mindig a saját oldaláról egy nem üres cellát kiválasztva kiveszi onnan az összes golyót, majd ezeket az óramutató járásával ellenkező irányba egyesével leteszi a következő cellákba és a saját bázisába (az ellenfél bázisát ki kell hagyni, de a celláit nem), amíg el nem fogy a kezéből az összes golyó.
- Ha az utolsó golyót a saját bázisába rakta, akkor újra ő következik.
- Ha az utolsó golyót a saját oldalán egy olyan cellába rakta, ahol azelőtt nem volt egy golyó sem, akkor azt az egy golyót, és az adott cellával szemközti cellában (ellenfél oldala) lévő összes golyót is a saját bázisába kell, hogy tegye.
- A játék akkor ér véget, ha a soron következő játékos oldalán mindegyik cella üres.
- Ekkor, ha a másik játékos oldalán még van golyó, akkor azt mindet a saját bázisába teheti.

- Az győz, akinek a játék végén több golyó van a bázisában.

1.2 Használati elképzelés

A játék során lehetőség lesz a játék fájlba mentésére, illetve a program indításakor ilyen fájl alapján történő játékbetöltésre, folytatásra.

A felhasználó számára a játékfelület úgy fog kinézni, hogy minden cellát egy egész szám fog ábrázolni (éppen mennyi golyó van benne) valamint mellette a gödör azonosítója (pl.: A, B, C…).

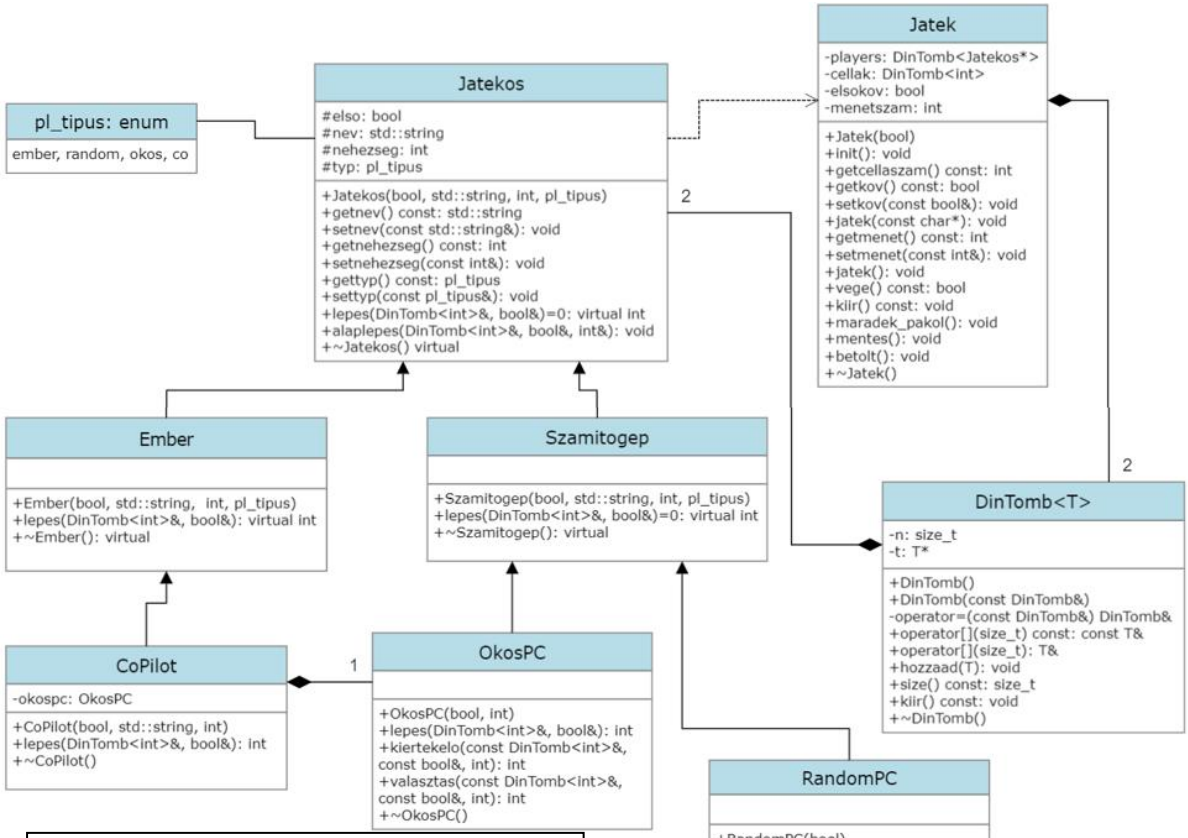
Ezek a gödrök két oszlopként lesznek a konzolra írva, és előtte, illetve utána a játékosok bázisának értéke. Emellett ki lesz írva, hogy melyik oldal melyik játékosé, ki következik és hogy hány lépés történt a játék kezdete óta.

A felhasználó úgy tudja kiválasztani a kiürítendő cellát, hogy beírja annak azonosítóját, ezután a lépések automatikusan megtörténnek.

2. Terv

A feladat elkészítéséhez szükség van nyolc osztály objektumra, valamint a tényleges játékot működtető főfüggvényre (tesztprogram).

2.1 A mancala játék objektumtervének UML diagramja



Módosítások:

Jatek::betolt módosult: const char* paraméterrel veszi át a fájl címét

Jatek-ban új tesztfüggvények (lásd később)

2.2 A program osztályainak és fő függvényeinek leírása

DinTomb<T> osztálysablon:

- Dinamikusan tárol tetszőleges mennyiségű paraméterként megadható típusú adatot.
- hozzaad(T adat) függvénnyel lehet új adatot hozzáadni
- van index operátora
- lekérhető az indexe
- Jatekos* típusra specializált destruktorral rendelkezik, hogy a dinamikusan foglalt tömb dinamikusan foglalt elemeit is felszabadítsa

Jatek osztály:

- tárolja az aktuális játék adatait: a játékosok adatait (lásd: Jatekos osztály), a tábla adatait (cellak: méret és értékek), hogy melyik játékos következik és hogy hány lépés telt el a játék kezdete óta.
- a szükséges adattagokhoz rendelkezik getter és setter függvényekkel
- init(): új játék létrehozásakor inicializálja a játékot, a szükséges adatok betöltésével
- betolt(): elmentett játékot tölt be. A fájlból beolvasott adatokkal inicializálja a már folyamatban lévő játékot.
- mentes(): amennyiben a felhasználó elmentené a játékot ez a függvény írja meg a szükséges fájlt az aktuális játék paraméterekkel
- vege(): a játék futása közben minden lépés után megnézi, hogy szükség van-e még lépésre, vagy véget ért a játék
- kiir(): kiírja az aktuális játékhelyzetet a standard kimenetre, a specifikációban leírt módon
- maradek_pakol(): amennyiben a játék végetért, a játékszabályban leírt módon a maradék golyókat a játékos bázisába kell tenni, ezt valósítja meg a függvény

Jatekos absztrakt alaposztály:

- Rendelkezik egy játékost meghatározó adatokkal:
    - elso: az adott játékos lett-e először hozzáadva. (Mivel mindkét játékos celláit egy tömbben tároljuk,

ezért fontos tudni, hogy a sajátja a tömb első fele vagy a második)
    - nev: a játékos neve (Ember és CoPilot esetén saját)
    - nehezseg: az okos algoritmus hány lépéssel gondolkozzon előre (OkosPC és CoPilot esetén fontos)
    - typ: milyen típusú játékosról van szó: saját enum lett létrehozva rá, a pl_tipus, ami rendre ember, random, okos vagy co lehet.
- a szükséges adattagokhoz rendelkezik getter és setter függvényekkel
- rendelkezik operator<< függvénnyel
- lepes(): tisztán virtuális függvény, így absztrakt alaposztályként viselkedik a Jatekos osztály objektum, nem lehet példányosítani, csak a leszármazottait akiknek meg lett valósítva a függvény.
- alaplepes(): a paraméterként megadott indexű cellát kiválasztva módosítja a paraméterként átadott játéktáblát a szabályoknak megfelelően, valamint azt is kezeli, hogy a soron következő játékos ki lesz. Ezt fogja használni az összes leszármazott lepes() függvénye miután az adott módszerrel eldöntötték, hogy melyik indexű cellát ürítik ki. Így a leszármazottak lepes() függvényében már csak a módszert részletezem.

Ember alosztály:

- a Jatekos osztály leszármazottja, egyéni nevet kér a felhasználótól
- lepes(): bekéri a felhasználótól, hogy melyik cellát kiválasztva szeretne lépni. A specifikációban megadott módon az azonosító karaktert kell megadnia, a függvény annak a helyességét ellenőrzi és konvertálja indexszé.

CoPilot alosztály:

- az Ember osztály leszármazottja, egyéni nevet és nehézséget is kér
- rendelkezik egy saját OkosPC típusú adattaggal, melynek a saját nehezseg-et és elso-t állítja be. Ennek segítségével

fog hozzáférni az OkosPC osztályban implementált okos algoritmushoz. (Azért nem annak a leszármazottja, mert az Ember lepes() függvényére is szüksége van és akkor meg azt kellene átadni neki.)
- lepes(): az OkosPC adattag tagfüggvényeivel elvégzi az okos algoritmust, mellyel megkapja ajavasolt indexet. Ezt azonosító karakterré konvertálja, közli a felhasználóval, majd elvégzi az Ember::lepes()-t

Szamitogep alosztály

- lepes(): tisztán virtuális, mivel Szamitogep példány nem létezhet, csak konkrétan OkosPC vagy RandomPC

RandomPC alosztály

- a Szamitogep osztály leszármazottja
- lepes(): index véletlen szám generálásával kiválaszt egy megfelelő cellát (nem nulla értékűt a saját oldaláról), majd a felhasználónak kiírja, hogy mi a választott cella és elvégzi a lépést (alaplepes())

OkosPC alosztály

- a Szamitogep osztály leszármazottja, meg kell adni a nehézségét
- lepes(): a kiertekelo() és valasztas() függvényekkel megvalósítja az okos algoritmust, melyet külön részben részletezek

További függvények:

- cim(): a program indításakor kiírja a játék nevét (de nem akárhogy)
- beolv_ell(const T&, const T&): T függvénysablon: bekér a felhasználótól egy T típusú adatot, addig amíg helyes bemenet nem lesz: az értéke a paraméterként megadottak közé esik. Ha helyes, visszatér vele (int-hez van csak használva, karakterhez is jó lehet, stringre specializálva: a szó max hosszát vizsgálja)

2.3 Okos algoritmus működése:

Az algoritmus adott d mélységig (lépésig) megnézi az összes lehetséges kimenetelt. Amennyiben n db cella van egy játékos oldalán ez n^d esetet jelent.

Ezt a minimax eljárással valósítom meg, mely végignézi az összes lehetséges kimenetelt, melyeknek minden lépését kiértékelve a számítógép számára legkedvezőbbet választja ki.

A kiértékelés egy lépés után történik, és az értéke az lesz, hogy mekkora a különbség a számítógép bázisának értéke és a másik játékosé között. Amennyiben az adott lépés a számítógépé volt, tehát az ő körében vagyunk, akkor ezeknek az értékeknek a maximuma adja a legjobb választást, ha pedig az ellenfél köre van, akkor a legkisebb számot kell választani, ugyanis feltételezzük, hogy az ellenfél is az algoritmusnak megfelelően okosan fog játszani.

A kiértékelést rekurzióval hajtom végre, ahol a leállási feltétel, hogy a nehézség 0 lett (ekkor csak egyszerűen kiértékeljük az előbb meghatározott módon), vagy esetleg akkor is, ha mi jönnénk, de már nem lehet mit lépni, a játék véget ért.

Az OkosPC::valasztas() hajtja végre az első n db kiértékelést, így ott már d- -gyel hívja meg, és visszatérési értékként keresi a legnagyobb értékűt (mert ezt csak akkor hívja meg amikor ő következik) és annak az indexét is, hiszen azt kell visszaadnia a lepes()-nek.

Az OkosPC::kiertekelo() a valasztas-hoz hasonlóan értékeli ki az újabb n db esetet és folytatja a rekurziót, viszont itt figyelni kell külön esetként, hogy nem mindig a számítógép következik abban a lépésben amit nézünk, az előbb említett két eset itt kerül külön megvalósításra. Továbbá a leállási feltételt is itt kell megvalósítani.

2.4 Tervezés önrevíziója

A tervezés óta tett módosítások:

- main.cpp-ben új függvények teszteléshez: init_test és teszt_main

- jatek.cpp-ben (és így a Játék osztályban) új tagfüggvények teszthez:
    add_test, file_test, lepes_veg_teszt és okosalgo_teszt függvények
- pl_tipus enumban „random” módosítása „buta”-ra, a JPorta ellenőrzése miatt
- Jatek osztály betolt függvénye: a betöltendő fájl nevét a program előre kéri be és paraméterként adja át, hogy tesztelhető lehessen, ne kelljen felhasználó hozzá

3. Megvalósítás

A feladathoz elkészült nyolc osztály és öt tesztprogram, amikkel a játék fő automatikus részei tesztelhetők. Az osztályok végleges függvényei az objektum tervnél látható. A tervezési lépéshez képest csak pár apró módosítás történ. Az osztályok a saját nevük által elnevezett .h (vagy .hpp) és .cpp fájlokban találhatóak. A tesztprogram fő függvénye és a játékot működtető függvény (JPorta miatt kommentezve) a main.cpp fájlban lett elkészítve. A tesztesetek megvalósítása többségében a jatek.cpp-ben történt.

3.1 Osztályok bemutatása

Minden osztálynak a deklarációja és inline függvényei az ”osztalynev.h” fájlban található, míg a többi függvény az ”osztalynev.cpp” fájlokban van. A tagfüggvények teljes megvalósítása megtalálható a Mellékletek fülnél.

3.1.1 DinTomb<T> osztály

A dinamikus tömb egy osztálysablon tetszőleges mennyiségű, tetszőleges típus dinamikus tárolásához. A programban int-ek dinamikus tárolásához használom, továbbá specializáltam arra is, hogy Játékos* típust is képes legyen tárolni, illetve a dinamikusan foglalt Játékosok felszabadításáért is felel.

Public Member Functions

- **DinTomb** ()
  Default konstruktor.

- const T & **operator[]** (size_t i) const
- T & **operator[]** (size_t i)
- void **hozzaad** (T)
  Új adat hozzáadása a dinamikus tömbhöz

- **DinTomb** (const **DinTomb** &dt)
- size_t **size** () const
- void **kiir** () const
  Adattároló minden adatának felsorolása.

- **~DinTomb** ()

Destruktor.

Detailed Description

template<class T>

class DinTomb< T >

DinTomb osztály sablon dinamikus tároláshoz

Parameters

T	adattípus

Constructor & Destructor Documentation

template<class T > DinTomb< T >::DinTomb (const DinTomb< T > & dt) [inline]

Másoló konstruktor Használata a programban csak int-re

Parameters

dt	- lemásolandó tároló

Új tároló létrehozása és adatok egyenkénti átadása

template<class T > DinTomb< T >::~DinTomb

Destruktor Felszabadítja a dinamikusan foglalt adattömböt

Member Function Documentation

template<class T > void DinTomb< T >::hozzaad (T adat)

Új adat hozzáadása a dinamikus tömbhöz

Eggyel több adatnak helyfoglalás, majd átmásolás az új adattal együtt

Kell a T-nek = operator (implicit jó) és a DinTomb-nek [] operator (megcsinálva)

Régiek felszabadítása és a pointer megfelelõ helyre mutatásának beállítása

Parameters

adat	- sablonnak megfelelő típusú tárolóhoz hozzáadandó adat

Returns

- Memóriafoglalási hiba esetén kivételt dob

Ha nem sikerült foglalni, automatikusan kivételt dob (std::bad_alloc)

template<class T > T & DinTomb< T >::operator[] (size_t i) [inline]

Indexelés konstans függvénye

Parameters

i	- index

Returns

- referencia az adott indexű elemre
- hibás indexérték esetén kivételt dob

template<class T > const T & DinTomb< T >::operator[] (size_t i) const [inline]

Indexelés

Parameters

i	- index

Returns

- referencia az adott indexű elemre
- hibás indexérték esetén kivételt dob

template<class T > size_t DinTomb< T >::size () const [inline]

Méret lekérése

Returns

- tároló mérete

The documentation for this class was generated from the following files:
dintomb.hpp, dintomb.cpp

3.1.2 Jatek osztály

Public Member Functions

- **Jatek** (bool elso=true)
  Konstruktor
- void **init** ()
  Új játék inicializálása felhasználó által
- int **getcellaszam** () const
- bool **getkov** () const
- void **setkov** (const bool &elso)
- int **getmenet** () const
- void **setmenet** (const int &menet)
- void **jatek** ()
  Játék fő menetének kezelése.
- bool **vege** () const
  Játék vége.
- void **kiir** () const
  Játék állásának (tábla) kirajzolása.
- void **maradek_pakol** ()
  Játék vége esetén a maradék golyók bázisba rakása.
- void **mentes** ()
  Játék fájlba mentése.
- void **betolt** (const char *)
  Elmentett játék fájlból betöltése.
- bool **add_test** ()
  Teszt 2: játék inicializálása, adatokkal feltöltése.
- bool **file_test** ()

Teszt 3: elmentett fájl betöltése.

- bool **lepes_veg_teszt** ()
  Teszt 4: alaplépés és játékvégi funkciók működése.

- bool **okosalgo_teszt** ()
  Teszt 5: okos számítógép algoritmusának ellenőrzésé egyszerű példán.

- **~Jatek** ()
  Destruktor.

Detailed Description:

Játék osztály

Tartalma:

1, A játékállás setterei, getterei

2, Játék lefolyását végző fő függvények

3, A program tesztelését végző függvények

Construktor & Destruktor Documentation

Jatek::Jatek (bool elso = `true`*)* `[inline]`

Konstruktor (a cellák és játékosok hozzáadása külön történik)

Parameters

elso	- az első játékos következzen-e

Member Funcion Documentation

*void Jatek::betolt (const char ** fajlnev*)*

Elmentett játék fájlból betöltése. Csak akkor hívódik meg, ha a paraméterként megadott nevű fájl tényleg létezik. Azért lett előre bekérve és paraméterrel átadva, hogy a teszteléskor használhassuk automatikusan- Elmentésnek megfelelő formátum beolvasása

Hibás fájlformátum esetén kivételt dob, ami kilép a programból

Hibás fájlformátum lehet rossz típusú adat, vagy nem megengedett értékű adat

nev	- a betöltendő fájl neve (.txt-vel)

Hány lépés telt el, ki következik és hány cella van összesen

Játékosok típusának, nevének és nehézségüknek beolvasása (az első játékos lett először beleírva)

Megfelelő típusú játékos létrehozása dinamikusan (felszabadítás a DinTomb feladata)

Cellák létrehozása, értéküknek megadása

int Jatek::getcellaszam () const [inline]

Cellaszám lekérdezése

Returns

- táblán levő cellák száma (bázissal, mindkét oldallal)

bool Jatek::getkov () const [inline]

Következõ játékos lekérdezése

Returns

- első következik-e

int Jatek::getmenet () const [inline]

Menetszám lekérdezése

Returns

- hány lépést tettek meg a játék kezdete óta

void Jatek::init ()

Új játék inicializálása felhasználó által

Hány cella és cellánként hány golyó (beolvasosablon-nal, hibakezelés is)

A tömb első eleme a bal játékos első cellája, az utolsó eleme a jobb játekos bázisa

cellák hozzáadása, értékkel feltöltése

A bázisokban kezdetben 0 golyó van

Játékosok adatainak bekérése

Típusnak megfelelően csak a hozzá szükséges adatok bekérése, és kért típusnak megfelelő játékos létrehozása

DinTomb dinamikusan tárolja a dinamikusan foglalt játékos pointerét (felszabadítás a DinTomb feladata)

void Jatek::jatek ()

Játék fő menetének kezelése.

Amíg nincs vége elvégezzük a soron következő játékos lépését

A soron következő játékos lép

Felhasználó esetén (Ember vagy CoPilot) van rá esély, hogy lépés helyett mentene vagy kilépne (ha igen elvégezzük az, kilépésnél rákérdezünk azért még egyszer)

Megjegyzés: exit-nél karakterenként olvas be, ezért, ha több karaktert (stringet) ír a felhasználó, az első előforduló y/n lesz elfogadva

Ha véget ér a játék elvégezzük a maradékok bázisba pakolását, kirajzoljuk még utoljára a táblát és összegezzük a játék eredményét győztest hirdetve.

void Jatek::kiir () const

Játék állásának (tábla) kirajzolása.

Megjelenés: a két játékos oldala (cellái) két oszlopként lesznek feltüntetve

A bal oldali oszlop az első játékosé, a jobb a másodiké

Az óramutató járásával ellentétesen haladva az első játékos bázisa van lent a két oszlop között (alá írva az első játékos adatai), a második játékos bázisa van fent a két oszlop között (felette a második játékos adatai) A játékosok minden cellájához tartozik egy azonosító (első játékosnak a golyószámtól balra, másodiknak jobbra) óramutató járásával ellentétesen első játékosnak (fentről lefele) A, B, C, ... a második játékosnak (lentről felfele) a, b, c, ... 15 a max cellaszám egy oldalon

szóval csak latin betűk lesznek és könnyen lehet őket konvertálni oda-vissza az indexekből. Továbbá fel van tüntetve, hogy hány lépést tettek meg a játék kezdete óta, és hogy ki a soron következő játékos

void Jatek::maradek_pakol ()

Játék vége esetén a maradék golyók bázisba rakása

A soron következő játékosnak biztos nem lesz maradék golyója, mert akkor nem ért volna véget

void Jatek::mentes ()

Játék fájlba mentése.

Ilyen formátumban lesz majd betöltve később

Fájlnév: "jatekos1neve"+"vs"+"jatekos2neve"+".txt"

void Jatek::setkov (const bool & elso) [inline]

Következő játékos beállítása

Parameters

elso	- az első következzen-e, ez lesz beállítva

void Jatek::setmenet (const int & menet) [inline]

Menetszám beállítása

Parameters

menet	- beállítja hány lépést tettek meg a játék kezdete óta

bool Jatek::vege () const

Játék vége

A soron következő játékos oldalát kell végig nézni, hogy van-e nem üres cella

Returns

véget ért-e a játék

The documentation for this class was generated from the following files:

jatek.h, jatek.cpp

3.1.3 Jatekos osztály

Absztrakt alaposztály, gyerekei: **Ember** és **Szamitogep**.

Public Member Functions

- **Jatekos** (bool first, std::string str="", int diff=0, pl_tipus mi=ember)
- bool **getelso** () const
- std::string **getnev** () const
- void **setnev** (const std::string &str)
- int **getnehezseg** () const
- void **setnehezseg** (const int &diff)
- pl_tipus **gettyp** () const
- void **settyp** (const pl_tipus &mi)
- virtual int **lepes** (**DinTomb**< int > &, bool &)=0
- void **alaplepes** (**DinTomb**< int > &, bool &, int &)
  Lépés végrehajtása.

- virtual **~Jatekos** ()
  Virtuális destruktor.

Protected Attributes

- bool **elso**
  Ő-e az első (az ő celláival kezdődik a tömb)
- std::string **nev**
  Név.

- int **nehezseg**
  Nehézség (okos algoritmus rekurziójának mélysége)

- pl_tipus **typ**
  Játékos típusa.

Construktor & Destructor Documentation

Jatekos::Jatekos (bool first, *std::string* str = **""**, *int* diff = *0*, *pl_tipus* mi = *ember)* `[inline]`

Konstruktor

Parameters

first	- első-e
str	- név, default: ""
diff	- nehézség, default: 0
mi	- játékos típusa, default: ember

Member Funcion Documentation

void Jatekos::alaplepes (DinTomb< int > & cellak, *bool &* kov, *int &* idx)

Lépés végrehajtása: Minden típus a végén egyformán lép, kiválasztja, hogy melyik cellát üríti ki

Parameters

cellak	- tábla celláinak tömbje (referencia - módosítja)
kov	- első játékos következik-e (a következő lépéshez fontos)
idx	- melyik cellát üríti ki a játékos

Ha a saját bázisukba került az utolsó, akkor újra õ következik (nem változik a kov értéke)

Ha a saját oldalukon egy lyukba fejezték be, ami eddig üres volt (most 1 az értéke) és azzal szemben nem üres lyuk van Átpakolják mindkét lyuk golyóit a bázisukba

A másik játékos következik (ha nem akkor ide már nem jut el)

bool Jatekos::getelso () const `[inline]`

Hányadik játékos

Returns

- igaz ha ő az első

int Jatekos::getnehezseg () const [inline]

Játékosnehézség lekérdezés

Returns

- nehézség

std::string Jatekos::getnev () const [inline]

Játékos névlekérdezés

Returns

- név

pl_tipus Jatekos::gettyp () const [inline]

Játékostípus lekérdezés

Returns

- típus

virtual int Jatekos::lepes (DinTomb< int > & , bool &) [pure virtual]

Lépés végrehajtása játékos típusától függően

Legyen absztrakt alaposztály (tisztán virtuális), így nem lehet simán Jatekos példányt létrehozni

Minden alosztálynak külön megvalósítva

Implementálva az **Ember**, **RandomPC**, **OkosPC**, **CoPilot** és **Szamitogep** alosztályokban.

void Jatekos::setnehezseg (const int & diff) [inline]

Játékosnehézség beállítása

Parameters

diff	beállítandó nehézség

void Jatekos::setnev (const std::string & str) [inline]

Játékos névbeállítás

Parameters

str	beállítandó név

void Jatekos::settyp (const pl_tipus & mi) [inline]

Játékostípus beállítása

Parameters

mi	beállítandó típus

The documentation for this class was generated from the following files:

jatekos.h, jatekos.cpp

3.1.4 Ember osztály

Inherits **Jatekos**.

Inherited by **CoPilot**.

Public Member Functions

- **Ember** (bool first, std::string str="jatekos", int diff=0, pl_tipus mi=ember)
- virtual int **lepes** (**DinTomb**< int > &, bool &)
  *Lépés végrehajtása **Ember** játékosnak.*

- virtual **~Ember** ()
  Virtuális destruktor (van leszármazottja)

Public Member Functions inherited from Jatekos

- **Jatekos** (bool first, std::string str="", int diff=0, pl_tipus mi=ember)
- bool **getelso** () const
- std::string **getnev** () const
- void **setnev** (const std::string &str)
- int **getnehezseg** () const
- void **setnehezseg** (const int &diff)
- pl_tipus **gettyp** () const
- void **settyp** (const pl_tipus &mi)
- virtual int **lepes** (**DinTomb**< int > &, bool &)=0
- void **alaplepes** (**DinTomb**< int > &, bool &, int &)
  Lépés végrehajtása.

- virtual **~Jatekos** ()
  Virtuális destruktor.

Additional Inherited Members

Protected Attributes inherited from Jatekos

- bool **elso**
  ő-e az első (az ő cellájával kezdődik a tömb)

- std::string **nev**
  Név.

- int **nehezseg**
  Nehézség (okos algoritmus rekurziójának mélysége)

- pl_tipus **typ**
  Játékos típusa.

Constructor & Destructor Documentation

Ember::Ember (bool first, std::string str = "jatekos", int diff = 0, pl_tipus mi = ember)[inline]

Konstruktor

Parameters

first	- első-e

str	- név, default: "jatekos"
diff	- nehézség, default: 0
mi	- játékos típusa, default: ember

Member Funcion Documentation

int Ember::lepes (DinTomb< int > & cellak, bool & kov) [virtual]

Lépés végrehajtása **Ember** játékosnak.

Parameters

cellak	- tábla celláinak tömbje
kov	- első játékos következik-e

Returns

- egyedül Embernél és CoPilotnél van jelentõssége
- funkció választása: 0 - lép, 1 - elmenti a játékot, 2 - kilép

A tábla kiíratásának megfelelően beolvas egy azonosító karaktert Hibakezelés: Ha hibás a karakter, nem fogadja el, újat kér

Mentés esetén return 1

Mentés nélküli kilépés esetén return 2

Nem megfelelő karaktert írt be, nincs ilyen azonosítójú lyuka

Nem választhat üres lyukat

Nem egy azonosító karatkert adott meg

Megfelelő index esetén végrehajtja a lépést (módosítja a cellákat, következő játékost is)

Lépés esetén return 0

Implements **Jatekos**

Reimplemented in **CoPilot**

The documentation for this class was generated from the following files:

jatekos.hjatekos.cpp

3.1.5 Szamitogep osztály

Inherits **Jatekos**.

Inherited by **OkosPC**, and **RandomPC**.

Public Member Functions

- **Szamitogep** (bool first, std::string str="", int diff=0, pl_tipus mi=buta)
- virtual int **lepes** (**DinTomb**< int > &, bool &)=0
- virtual **~Szamitogep** ()
  Virtuális destruktor (van leszármazottja)

Public Member Functions inherited from Jatekos

- **Jatekos** (bool first, std::string str="", int diff=0, pl_tipus mi=ember)
- bool **getelso** () const
- std::string **getnev** () const
- void **setnev** (const std::string &str)
- int **getnehezseg** () const
- void **setnehezseg** (const int &diff)
- pl_tipus **gettyp** () const
- void **settyp** (const pl_tipus &mi)
- virtual int **lepes** (**DinTomb**< int > &, bool &)=0
- void **alaplepes** (**DinTomb**< int > &, bool &, int &)
  Lépés végrehajtása.

- virtual **~Jatekos** ()
  Virtuális destruktor.

Additional Inherited Members

Protected Attributes inherited from Jatekos

- bool **elso**
  ő-e az első (az ő cellájával kezdődik a tömb)

- std::string **nev**
  Név.

- int **nehezseg**
  Nehézség (okos algoritmus rekurziójának mélysége)

- pl_tipus **typ**
  Játékos típusa.

Detailed Description

Számítógép alosztály, olyan játékos típusokhoz, amik felhasználó nélkül, maguktól működnek

---

Constructor & Destructor Documentation

Szamitogep::Szamitogep (bool first, std::string str = "", int diff = 0, pl_tipus mi = buta)[inline]

Konstruktor

Parameters

first	- első-e
str	- név, default: ""
diff	- nehézség, default: 0
mi	- játékos típusa, default: buta

Member Function Documentation

virtual int Szamitogep::lepes (DinTomb< int > & , bool &) [pure virtual]

Lépés végrehajtása tisztán virtuális, hogy absztrakt osztály legyen. Nincs **Szamitogep** játékos, csak annak a leszármazottjai

Implements **Jatekos**

Implemented in **RandomPC** and **OkosPC**

The documentation for this class was generated from the following file: ***jatekos.h***

3.1.6 RandomPC osztály

Inherits **Szamitogep**.

Public Member Functions

- **RandomPC** (bool first)
- int **lepes** (**DinTomb**< int > &, bool &)
  Lépés végrehajtása.

- **~RandomPC** ()
  Destruktor.

Public Member Functions inherited from Szamitogep

- **Szamitogep** (bool first, std::string str="", int diff=0, pl_tipus mi=buta)
- virtual int **lepes** (**DinTomb**< int > &, bool &)=0
- virtual **~Szamitogep** ()
  Virtuális destruktor (van leszármazottja)

Public Member Functions inherited from Jatekos

- **Jatekos** (bool first, std::string str="", int diff=0, pl_tipus mi=ember)
- bool **getelso** () const
- std::string **getnev** () const
- void **setnev** (const std::string &str)
- int **getnehezseg** () const
- void **setnehezseg** (const int &diff)
- pl_tipus **gettyp** () const
- void **settyp** (const pl_tipus &mi)
- virtual int **lepes** (**DinTomb**< int > &, bool &)=0
- void **alaplepes** (**DinTomb**< int > &, bool &, int &)
  Lépés végrehajtása.

- virtual **~Jatekos** ()
  Virtuális destruktor.

Additional Inherited Members

Protected Attributes inherited from Jatekos

- bool **elso**
  ő-e az első (az ő cellájával kezdődik a tömb)

- std::string **nev**
  Név.

- int **nehezseg**
  Nehézség (okos algoritmus rekurziójának mélysége)

- pl_tipus **typ**
  Játékos típusa.

Detailed Description

RandomPC alosztály. Olyan játékos, aki mindig véletlenszerűen választ a lehetséges megengedett lépései közül

---

Constructor & Destructor Documentation

RandomPC::RandomPC (bool first) [inline]

Konstruktor

Parameters

first	- első-e a többi paraméterét tudjuk (típus adott, név automatikus, nehézség irreleváns)

Member Funcion Documentation

int RandomPC::lepes (DinTomb< int > & cellak, bool & kov) [virtual]

Lépés végrehajtása.

Lépés végrehajtása **RandomPC** játékosnak

Parameters

cellak	- tábla celláinak tömbje
kov	- első játékos következik-e

Returns

0: egyetlen funkció a lépés

Attól függően, hogy hányadik játékos a megfelelő indexre állítja Ha üres cellát választ újra választania kell

Megfelelő index esetén végrehajtja a lépést (módosítja a cellákat, következőjátékost is)

Implements **Szamitogep**

---

*The documentation for this class was generated from the following files: **jatekos.h, jatekos.cpp***

3.1.7 OkosPC osztály

Inherits **Szamitogep**.

Public Member Functions

- **OkosPC** (bool first, int diff=1)
- int **lepes** (**DinTomb**< int > &, bool &)
  Lépés végrehajtása.

- int **kiertekelo** (const **DinTomb**< int > &, const bool &, int)
- int **valasztas** (const **DinTomb**< int > &, const bool &, int)
- **~OkosPC** ()
  Destruktor.

Public Member Functions inherited from Szamitogep

- **Szamitogep** (bool first, std::string str="", int diff=0, pl_tipus mi=buta)
- virtual int **lepes** (**DinTomb**< int > &, bool &)=0
- virtual **~Szamitogep** ()
  Virtuális destruktor (van leszármazottja)

Public Member Functions inherited from Jatekos

- **Jatekos** (bool first, std::string str="", int diff=0, pl_tipus mi=ember)
- bool **getelso** () const
- std::string **getnev** () const
- void **setnev** (const std::string &str)
- int **getnehezseg** () const
- void **setnehezseg** (const int &diff)
- pl_tipus **gettyp** () const
- void **settyp** (const pl_tipus &mi)
- virtual int **lepes** (**DinTomb**< int > &, bool &)=0
- void **alaplepes** (**DinTomb**< int > &, bool &, int &)
  Lépés végrehajtása.

- virtual **~Jatekos** ()
  Virtuális destruktor.

Additional Inherited Members

Protected Attributes inherited from Jatekos

- bool **elso**
  ő-e az első (az ő cellájával kezdődik-e a tömb)

- std::string **nev**
  Név.

- int **nehezseg**
  Nehézség (okos algoritmus rekurziójának mélysége)

- pl_tipus **typ**
  Játékos típusa.

Detailed Description

OkosPC alosztály. A megadott nehézségnek megfelelően előre megírt algoritmussal kiszámolja a számára legkedvezőbb lépést

---

Constructor & Destructor Documentation

OkosPC::OkosPC (bool first, int diff = 1) [inline]

Konstruktor

Parameters

first	- első-e
diff	- nehézség, default: 1 A többi paraméterét tudjuk (típus adott, név automatikus)

Member function Documentation

int OkosPC::kiertekelo (const DinTomb< int > & cellak, const bool & kov, int diff)

Egy aktuális lépés sikerességét számolja ki. Ez hívódik meg rekurzióval

Kiértékelő: értékeli az aktuális táblát a számítógép szemszögéből a bázisok segítségével, rekurzióval Működése a valasztas fv-hez hasonló, DE

- A választás meghívásakor biztos a számítógép következik, míg itt változó, ezért: Esetszétválasztást kell végezni, mert ha az ellenfél jön a számára legkedvezőbb esettel kell számolni
- A végleges index meghatározása nem ennek a függvénynek a felelőssége
- Parameters

cellak	- tábla celláinak tömbje
kov	- első játékos következik-e
diff	- játékos nehézsége: hány lépéssel számoljon előre

- Returns

  Az aktuális lépésig mekkora értékű ez az opció, mennyire kedvező

  Végignézi az összes lehetséges kimenetet. Csak az egyik oldal celláira kell (azzal tud lépni)

Kilépési feltételek:

1, Ha már véget ért a játék, akkor nincs mit tovább nézni

2, Ha elérte a diff==0-t, vagyis eljutott a diff-edik lépésig amíg előre kellett számolnia.

Ha a számítógép jön számára a lehető legjobb kimenetelt kell nézni

Végignézi az összes választási lehetőségének kimenetelét

Megnézi, hogy ha ezt választaná milyen játékállás lenne

Ezt a játékállást is kiértékeli, úgy, hogy megnézi az összes lehetőséget, de már eggyel kevesebb mélységig

Lehető legjobb érték lett-e, ha nem a számítógép jön, akkor a számítógép számára lehető legrosszabb kimenetelt kell nézni

A játékot az ellenfél szemszögéből kell nézni, viszont az õ függvényeihez nem férünk hozzá Az alaplepest a számítógépre hívjuk meg A számítógép elsőjét fogja figyelembe venni Ez megoldható úgy, hogy módosítjuk az elso értékét, mintha a számítógép a másik oldalon játszana (például így jó bázist fog kihagyni a golyók elhelyezésekor) Erre a cellák végignézésekor és az alaplépéskor szükség van, DE Utána a kiértékeléshez (lehető legrosszabb eset), vissza kell állítani (hogy a következő lépés vizsgálatakor is jól működjön, és majd max ott is megcseréljük)

Most pont akkor kell hozzáadni (hogy a másik oldalt nézze), ha ő az első

Végignézi az összes választási lehetőségének kimenetelét

Aktuális kiválasztásával mi történik

Vissza kell állítani, hogy a kiértékelést már jól csinálja

Lehető legrosszabb érték-e

int OkosPC::lepes (DinTomb< int > & cellak, bool & kov) [virtual]

Lépés végrehajtása **OkosPC** játékosnak

Parameters

cellak	- tábla celláinak tömbje
kov	- első játékos következik-e

Returns

0: egyetlen funkció a lépés

Végrehajtja az algoritmust és visszatér a választott cella indexével Nem referenciával adja át, így nem módosítja a tényleges játékállást

Végrehajtja a lépést (módosítja a cellákat, következő játékost is)

Implements **Szamitogep**

int OkosPC::valasztas (const DinTomb< int > & cellak, const bool & kov, int diff)

Visszatér az algoritmus által javasolt cellaindexszel Innen indul a rekurzió

Választás: kiválasztja a számítógép szemszögéből legkedvezőbb lyukat amivel lép

Parameters

cellak	- tábla celláinak tömbje
kov	- első játékos következik-e
diff	- játékos nehézsége: hány lépéssel számoljon előre

Returns

- Az algoritmus szerinti legkedvezőbb lépéssel, a kiürítendő cella indexével tér vissza

Végignézi az összes lehetséges kimenetet Csak a saját celláira kell (azzal tud lépni)

Elsőnek mindig õ következik: ilyenkor a legjobb kimenetelt kell nézni

Megnézzük, hogy az aktuális cellát választva milyen játékállás lesz

Ezt az új állást kiértékeljük úgy, hogy már csak eggyel kevesebb mélységig kell vizsgálnia

A lehető legjobb esetet kell nézni, ezért, ha ez az, el is mentjük az indexet (és az értéket is a későbbi összehasonlításhoz)

The documentation for this class was generated from the following files:
jatekos.hjatekos.cpp

3.1.8 CoPilot osztály

Inherits **Ember**. Contains **OkosPC**

Public Member Functions

- **CoPilot** (bool first, std::string str="copilot", int diff=1)
- int **lepes** (**DinTomb**< int > &, bool &)
- **~CoPilot** ()
  Destruktor.

Public Member Functions inherited from Ember

- **Ember** (bool first, std::string str="jatekos", int diff=0, pl_tipus mi=ember)
- virtual int **lepes** (**DinTomb**< int > &, bool &)
  *Lépés végrehajtása **Ember** játékosnak.*

- virtual **~Ember** ()
  Virtuális destruktor (van leszármazottja)

Public Member Functions inherited from Jatekos

- **Jatekos** (bool first, std::string str="", int diff=0, pl_tipus mi=ember)
- bool **getelso** () const
- std::string **getnev** () const
- void **setnev** (const std::string &str)
- int **getnehezseg** () const
- void **setnehezseg** (const int &diff)
- pl_tipus **gettyp** () const
- void **settyp** (const pl_tipus &mi)
- virtual int **lepes** (**DinTomb**< int > &, bool &)=0
- void **alaplepes** (**DinTomb**< int > &, bool &, int &)
  Lépés végrehajtása.

- virtual **~Jatekos** ()
  Virtuális destruktor.

Additional Inherited Members

Protected Attributes inherited from Jatekos

- bool **elso**
  ő-e az első

- std::string **nev**
  Név.

- int **nehezseg**
  Nehézség (okos algoritmus rekurziójának mélysége)

- pl_tipus **typ**
  Játékos típusa.

Detailed Description

CoPilot alosztály (**Ember** és **OkosPC** ötvözete) Amellett, hogy emberként viselkedik, így a felhasználó dönti el mit lép. Segít neki egy megadott nehézségű OkosPC, ami egy általa ideálisnak gondolt javaslatot ad a felhasználónak

Private Attributes

- OkosPC **okospc**
  Mivel az okos algoritmust elvégző függvények az OkosPC függvényei,
  a CoPilot pedig az Ember alosztálya, ezért szükség van egy OkosPC adattagra.
  Rajta keresztül végrehajthatjuk a co-pilot OkosPC részét is

Constructor & Destructor Documentation

CoPilot::CoPilot (bool first, std::string str = "copilot", int diff = 1) [inline]

Konstruktor

Parameters

first	- első-e
str	- név, default: "copilot"
diff	- nehézség, default: 1

Member Function Documentation

int CoPilot::lepes (DinTomb< int > & cellak, bool & kov) [virtual]

Lépés végrehajtása okospc lépése (adattaggal) majd az ember lépése (kompatibilitás)

Lépés végrehajtása **CoPilot** játékosnak

Parameters

cellak	- tábla celláinak tömbje
kov	- első játékos következik-e

Returns

- egyedül Embernél és CoPilotnél van jelentősége

- funkció választása: 0 - lép, 1 - elmenti a játékot, 2 - kilép

Először végrehajtja az okos algoritmust (az adattagjának segítségével)

Az algoritmus által javasolt cellát a felhasználó tudtára adja

Majd végrehajtja az **Ember** osztály lépését, ahol az ember dönt melyik cellát választja

Reimplemented from **Ember**

The documentation for this class was generated from the following files:
jatekos.hjatekos.cpp

3.2 Fájldokumentáció

3.2.1 beolvasosablon.cpp

Reference

```
#include "memtrace.h"
#include "beolvasosablon.hpp"
```

Functions

- std::string **beolv_ell_str** (const size_t &maxmeret)
  Stringre működő ellenőrző beolvasás.

Detailed Description

beolvasosablon.cpp beolv_ell_str függvényének megvalósítására

---

Function Documentation

std::string beolv_ell_str (const size_t & maxmeret)

Stringre működő ellenőrző beolvasás A megadható határ a string hossza

Parameters

maxmeret	- beolvasott string maximális hossza

3.2.2 beolvasosablon.hpp

File Reference

```
#include "memtrace.h"
#include <iostream>
#include <limits>
```

Functions

- template<typename T > T **beolv_ell** (const T &kezd, const T &veg)
- std::string **beolv_ell_str** (const size_t &maxmeret)
  Stringre működő ellenőrző beolvasás.

Detailed Description

Függvénysablon tetszőleges típusú adatok beolvasására

Cél: ne legyen sok ismétlés a kódban (ezért függvény) és hogy ne csak egy valamire működjön (sablon, bár most sok mindenre nem fogjuk használni)

Function Documentation

template<typename T > T beolv_ell (const T & kezd, const T & veg)

Sablon egy felhasználó által megadott adat beolvasásához (char és int most) Addig kér be adatot, amíg nem ad egy megfelelőt a felhasználó

Parameters

kezd	- legalább ennyi legyen az értéke
veg	- maximum ennyi legyen az értéke

Returns

- a felhasználó által megadott megfelelő adat

std::string beolv_ell_str (const size_t & maxmeret)

Stringre működő ellenőrző beolvasás A megadható határ a string hossza

Parameters

maxmeret	- beolvasott string maximális hossza

beolvasosablon.hpp

```
8 #ifndef BEOLVASOSABLON_HPP_INCLUDED
9 #define BEOLVASOSABLON_HPP_INCLUDED
10
11 #include "memtrace.h"
12 #include <iostream>
13 #include <limits>
14
20 template<typename T>
21 T beolv_ell(const T& kezd, const T& veg) {
22     bool hibas=true;
23     T kapott;
24     do {
25         if(std::cin>>kapott&&kapott>=kezd&&kapott<=veg) hibas=false;
26         else {
27             //Szöveg esetén sem kerül végtelen ciklusba, hanem új inputot vár
28             std::cout<<"Hibas bemenet, kerem probalja ujra.\n";
29             std::cin.clear();
30             std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
31         }
32     } while(hibas);
33     return kapott;
34 }
35
37 std::string beolv_ell_str(const size_t& maxmeret);
38
39 #endif // BEOLVASOSABLON_HPP_INCLUDED
```

3.2.3 dintomb.cpp

Reference

```
#include <iostream>
#include <cstring>
#include "memtrace.h"
#include "dintomb.hpp"
#include "jatekos.h"
```

Functions

- void **teszt** ()
  *Teszt a **DinTomb** függvényeinek helyes működésére.*

Detailed Description

DinTomb sablon osztály és specializált függvényeinek megvalósítása

3.2.4 dintomb.hpp

File Reference

```
#include "memtrace.h"
```

Classes

class **DinTomb< T >**

Detailed Description

DinTomb sablon tetszőleges mennyiségű, tetszőleges típus dinamikus tárolásához Sablon specializációja: dinamikusan foglalt játékosok pointereinek tárolására

dintomb.hpp

```
7 #ifndef DINTOMB_HPP_INCLUDED
8 #define DINTOMB_HPP_INCLUDED
9
10 #include "memtrace.h"
11
14 template <class T>
15 class DinTomb {
16     size_t n;
17     T* t;
18     DinTomb& operator=(const DinTomb& dt); //kívülről nem hozzáférhető
19 public:
20
22     DinTomb(): n(0) {
23         t=NULL;
24     }
25
30     const T& operator[](size_t i) const {
31         if(i<n&&i>=0) return t[i];
32         else throw(std::out_of_range("Indexhiba!"));
33     }
34
39     T& operator[](size_t i) {
40         if(i<n&&i>=0) return t[i];
41         else throw(std::out_of_range("Indexhiba!"));
42     }
43
45     void hozzaad(T);
46
50     DinTomb(const DinTomb& dt): n(0) {
51         t=NULL;
53         for(size_t i=0; i<dt.n; i++) {
54             hozzaad(dt[i]);
55         }
56     }
57
60     size_t size() const { return n; }
61
63     void kiir() const;
64
66     ~DinTomb();
67 };
68 #endif // DINTOMB_HPP_INCLUDED
```

3.2.5 jatek.cpp

File Reference

```
#include <ctype.h>
#include <fstream>
#include <iomanip>
#include "memtrace.h"
#include "jatek.h"
#include "beolvasosablon.hpp"
```

Detailed Description

Játék osztály függvényeinek definiálása (teszteket kivéve)

Minden függvény az osztályoknál részletezve

3.2.6 jatek.h

File Reference

```
#include "memtrace.h"
#include "jatekos.h"
```

Classes

class **Jatek**

Detailed Description

Játék osztály deklarálása és inline függvényei

Minden függvény az osztályoknál részletezve

jatek.h

```
6 #ifndef JATEK_H_INCLUDED
7 #define JATEK_H_INCLUDED
8
9 #include "memtrace.h"
10 #include "jatekos.h"
11
17 class Jatek {
18     DinTomb<Jatekos*> players;
19     DinTomb<int> cellak;
20     bool elsokov;
21     int menetszam;
22 public:
25     Jatek(bool elso=true): elsokov(elso), menetszam(0) {}
26
28     void init();
29
32     int getcellaszam() const { return cellak.size(); }
33
36     bool getkov() const { return elsokov; }
37
40     void setkov(const bool& elso) { elsokov=elso; }
41
44     int getmenet() const { return menetszam; }
45
48     void setmenet(const int& menet) { menetszam=menet; }
49
51     void jatek();
52
54     bool vege() const;
55
57     void kiir() const;
58
60     void maradek_pakol();
61
63     void mentes();
64
66     void betolt(const char*);
67
69     bool add_test();
70
72     bool file_test();
73
75     bool lepes_veg_teszt();
76
78     bool okosalgo_teszt();
79
81     ~Jatek() {}
82 };
83
84 #endif // JATEK_H_INCLUDED
```

3.2.7 jatekos.cpp

File Reference

```
#include <cstring>
#include <cmath>
#include <ctime>
#include "memtrace.h"
#include "jatekos.h"
#include "jatek.h"
```

Functions

- std::ostream & **operator<<** (std::ostream &os, const **Jatekos** &pl)
  *Kiíró operátor (**DinTomb** kiir használata esetén)*

Detailed Description

Minden Játékosból származó függvény definiálása
A legtöbb az osztályok tagfüggvényei, korábban részletezve

Function Documentation

std::ostream & operator<< (std::ostream & os, const Jatekos & pl)

Kiíró operátor (**DinTomb** kiir használata esetén)

Parameters

os	- ostream típusú objektum
pl	- Játékos akinek az adatait kiírjuk

Returns

os

3.2.8 jatekos.h

File Reference

```
#include <iostream>
#include "memtrace.h"
#include "dintomb.hpp"
```

Classes

- class **Jatekos**
- class **Ember**
- class **Szamitogep**
- class **RandomPC**
- class **OkosPC**
- class **CoPilot**

Enumerations

- enum **pl_tipus** { **ember**, **buta**, **okos**, **co** }

Functions

- std::ostream & **operator<<** (std::ostream &, const **Jatekos** &)

*Kiíró operátor (**DinTomb** kiir használata esetén)*

Detailed Description

Játékos ősosztály és gyerekeinek deklarálása, inline függvényeik

---

Function Documentation

std::ostream & operator<< (std::ostream & os, const Jatekos & pl)

Kiíró operátor (**DinTomb** kiir használata esetén)

Parameters

os	- ostream típusú objektum
pl	- Játékos akinek az adatait kiírjuk

Returns

os

jatekos.h

```
6 #ifndef JATEKOS_H_INCLUDED
7 #define JATEKOS_H_INCLUDED
8
9 #include <iostream>
10 #include "memtrace.h"
11 #include "dintomb.hpp"
12
14 enum pl_tipus {
15     ember, buta, okos, co
16 };
17
19 class Jatekos {
20 protected:
21     bool elso;
22     std::string nev;
23     int nehezseg;
24     pl_tipus typ;
25 public:
31     Jatekos(bool first, std::string str="", int diff=0, pl_tipus mi=ember): elso(first),
nev(str), nehezseg(diff), typ(mi) {}
32
35     bool getelso() const {return elso; }
36
39     std::string getnev() const { return nev; }
40
43     void setnev(const std::string& str) { nev=str; }
44
47     int getnehezseg() const { return nehezseg; }
48
51     void setnehezseg(const int& diff) { nehezseg=diff; }
52
55     pl_tipus gettyp() const { return typ; }
56
59     void settyp(const pl_tipus& mi) { typ=mi; }
60
65     virtual int lepes(DinTomb<int>&, bool&)=0;
66
68     void alaplepes(DinTomb<int>&, bool&, int&);
69
71     virtual ~Jatekos() {}
72 };
73
75 std::ostream& operator<<(std::ostream&, const Jatekos&);
76
```

```
79 class Ember: public Jatekos {
80 public:
86     Ember(bool first, std::string str="jatekos", int diff=0, pl_tipus mi=ember):
Jatekos(first, str, diff, mi) {}
87
89     virtual int lepes(DinTomb<int>&, bool&);
90
92     virtual ~Ember() {}
93 };
94
97 class Szamitogep: public Jatekos {
98 public:
104     Szamitogep(bool first, std::string str="", int diff=0, pl_tipus mi=buta):
Jatekos(first, str, diff, mi) {}
105
109     virtual int lepes(DinTomb<int>&, bool&)=0;
110
112     virtual ~Szamitogep() {}
113 };
114
117 class RandomPC: public Szamitogep {
118 public:
122     RandomPC(bool first): Szamitogep(first, "randompc") {}
123
125     int lepes(DinTomb<int>&, bool&);
126
128     ~RandomPC() {}
129 };
130
133 class OkosPC: public Szamitogep {
134 public:
139     OkosPC(bool first, int diff=1): Szamitogep(first, "okospc", diff, okos) {}
140
142     int lepes(DinTomb<int>&, bool&);
143
146     int kiertekelo(const DinTomb<int>&, const bool&, int);
147
150     int valasztas(const DinTomb<int>&, const bool&, int);
151
153     ~OkosPC() {}
154 };
155
159 class CoPilot: public Ember {
160     OkosPC okospc;
163 public:
168     CoPilot(bool first, std::string str="copilot", int diff=1): Ember(first, str, diff,
co), okospc(first, diff) {}
169
172     int lepes(DinTomb<int>&, bool&);
173
175     ~CoPilot() {}
176 };
177 #endif // JATEKOS_H_INCLUDED
```

3.2.9 main.cpp

File Reference

```
#include <cstring>
#include <cmath>
#include <ctime>
#include <iomanip>
#include <fstream>
#include "memtrace.h"
#include "jatek.h"
#include "beolvasosablon.hpp"
```

Functions

- void **cim** ()
  Kezdőrajz: mancala

- bool **init_test** ()
- int **teszt_main** ()
- int **main** ()

Detailed Description

Felhasználói felülettel megvalósított főprogram a mancala játék tényleges használatához

A felhasználó válaszát értelemszerűen adja, az angol ábécé karaktereit használja, ékezeteket nem

---

Function Documentation

bool init_test ()

Teszt 1: Játék példány létrehozása

Returns

sikeres/sikertelen teszt

int main ()

Játékot végrehajtó főprogram
Főmenüjében kiválasztható a játék elkezdésének módja (új/betöltés)
Elindítja a játékot
A kommentezés a JPorta miatt van, mivel ennek a programrésznek szüksége van egy felhasználóra
Kipróbáláshoz/játszáshoz ki kell kommentezni

int teszt_main ()

Tesztelést végző főprogram

Meghívja az összes elkészített tesztprogramot és összegzi az eredményt

Mivel a cellak és players tárolók privátak, így a tagfüggvényeik is

Ezért a szükséges teszteseteknek a megfelelő osztályban (Jatek) hoztam létre a függvényeit

4. Tesztelés

A tesztesetek az alapján készültek el, hogy be tudja mutatni a program összes olyan funkcióját, ami nem kerül kapcsolatba a külvilággal (nincs szüksége felhasználó inputjára). Ezért hoztam létre öt darab tesztet. Ezek az első kivételével a Jatek osztály függvényeiként lettek megvalósítva, mivel az ellenőrzéshez szükség van a játékosok és cellák privát adatainak hozzáféréséhez.

Lefedettség: Mivel nem nyilvántartás jellegű programról van szó, hanem egy játékról, így kicsit korlátozottabbak voltak a lehetőségek arra, hogy minél több függvény működését jól tesztelhessem. A JPortán látható test coverage ezért is alacsony. Továbbá maguk a tesztek is úgy lettek kivitelezve, hogy feltétellel vizsgálom a helyes működést, és a hibát (ami sok szöveg kiírása a standard outputra) csak akkor jeleníti meg, ha az megtörténik. Tehát a program helyes működése esetén maguknak a tesztfüggvényeknek is csak kis része fut le. A felhasználói input ellenőrzésére teszt nem készült, viszont a hibakezelés egyszerűen, magától értetődően van megoldva (beolvasáskor rögtön ellenőrzi a helyességet és ha hibás (vagy típusa, vagy az értéke), akkor újra bekéri, amíg helyes bemenetet nem kap)

4.0 main_teszt

A main.cpp-ben található.

A JPorta számára ez lehetne a főprogram (main.cpp), viszont mivel ez egy játék, aminek fő célja, hogy játszanak vele, inkább létrehoztam ezt a függvényt, amit a main függvény meghív. A JPorta ellenőrzése miatt a main függvény többi része kommentelve lett, viszont, ha valaki játszani szeretne a kommentek kitörlésével ez rögtön működik.

Meghívja a teszteket, és figyeli, hogy melyik volt esetleg sikertelen. Az összes teszteset lefutása után összegzi az eredményt.

Egyszerűen bővíthető

4.1 init_test

A main.cpp-ben található

Játék példány létrehozása

Ellenőrzi, hogy létrejön-e a játék

4.2 add_test

A létrehozott játékot fel is tölti adatokkal.

Főképp a DinTomb illetve a Játékos osztályok működését ellenőrzi

Jó lett-e a méret, és a hozzáadott adatoknak az értéke is megegyezik.

A teszt ellenőrzi az indextúllépés esetét is. Ki kell írnia a standard outputra, hogy elkapta a kivételt.

Kivételkezelés elég csak a tesztprogramban, mert a játék során minden inputra történik ellenőrzés, maga a program meg jól lett megírva és nem indexel túl

4.3 file_test

Egy korábbi játék során elmentett fájl betöltését teszteli.

Sikerül-e ténylegesen betölteni az adatokat, hogy onnan folytatni lehessen a játékot.

A sikeres betöltés után el is menti a játékot, ezzel felülírva a fájlt. Így ha újra betöltenénk és hibásan működne, akkor tudjuk, hogy a mentés függvény működése hibás.

hiba.txt hibás formátumú fájl beolvasásával teszteljük a kivételkezelést. Ha helyesen működik, az inputon jeleznie kell, hogy detektálta.

4.4 lepes_veg_teszt

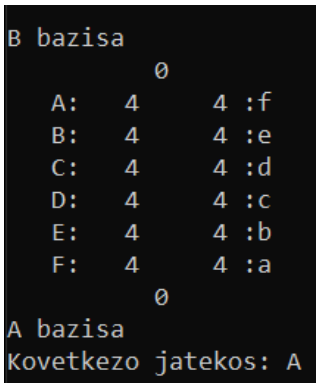
Alaplépés és játékvégi funkciók tesztelése

Ez a teszt a 3. tesztesetben betöltött játékban lép (csak nem a játékosok lépnek, hanem a teszt automatikusan)

Ellenőrzi a golyók megfelelő elhelyezését, és minden játékszabály helyes működését

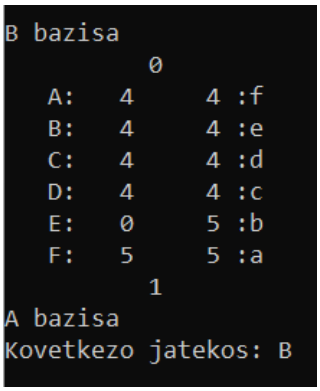
Üres cellát és bázis nem lehet választani, de ezt a főprogram nem is engedi, a hibakezelés ott van megoldva

Kezdőállapot



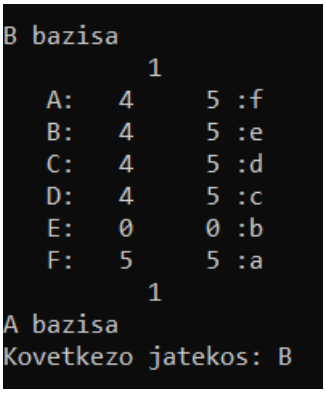
E választása

1. lépés után



b választása

2. lépés után



a választása

3. lépés után

```
B bazisa
         1
   A:    4      6 :f
   B:    4      6 :e
   C:    4      6 :d
   D:    4      6 :c
   E:    0      1 :b
   F:    5      0 :a
         1
A bazisa
Kovetkezo jatekos: A
```

4. lépés után

```
B bazisa
         1
   A:    0      6 :f
   B:    5      6 :e
   C:    5      6 :d
   D:    5      6 :c
   E:    0      0 :b
   F:    5      0 :a
         3
A bazisa
Kovetkezo jatekos: B
```

A választása

Az előbb a tesztben lefuttatott lépések láthatóak.

2. lépésben B játékos a bázisában fejezte be, ezért helyesen ő következett újra a 3. lépésnél. (ez a többinél helyesen nem fordult elő)

4. lépésben A játékos egy saját üres cellában fejezte be, amivel szemben B játékos cellájában is volt golyó, ezért helyesen megkapta mindkettő cella tartalmát a saját bázisába

Ezután leellenőriztem, hogy véget ért-e a játék ezen, és a vege_teszt.txt fájlból beolvasott játékon is. Az utóbbiban azt kapjuk, hogy vége lett, ezért a maradékok pakolását is végrehajtjuk, ezzel tesztelve azt is.

vege_teszt.txt befejezése:

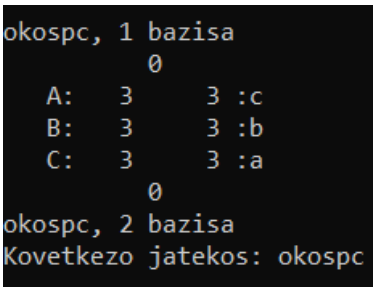
```
        JATEK
Eddigi lepesek szama: 22

B bazisa
         12
   A:    0      0 :d
   B:    1      0 :c
   C:    2      0 :b
   D:    0      0 :a
         17
A, 3 bazisa
Kovetkezo jatekos: B
```

```
B bazisa
         12
   A:    0      0 :d
   B:    0      0 :c
   C:    0      0 :b
   D:    0      0 :a
         20
A, 3 bazisa
Kovetkezo jatekos: B
```

4.5 okosalgo_teszt

Okos számítógép algoritmusának ellenőrzése egyszerű példán

Ez a teszt a 2. tesztesetben elkészített játékban számolja ki az aktuális játékos ideális lépést. Az első két létrehozott játékossal játszatjuk a játékot, akik mindketten OkosPC-k, az első nehézsége 2, a másodiké (1). Mivel a játéktábla is kisebb így nem kell sok esetet végignézni, és fejben is ki lehet számolni mi lesz az algoritmus szerinti legkedvezőbb lépés.



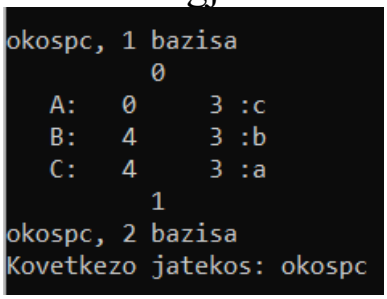
1. lépéshez

első játékos okos, 2 nehézséggel, ő következik

ha nem A-t választja akkor nem ő jön újra, így két lépés alatt legjobb esetben 0 lesz a két bázis közti különbség

ha A-t választja újra ő jön és így két lépés alatt legrosszabb esetben is 2-vel több lesz az ő bázisában

→ a A-t fogja választani



2. lépéshez

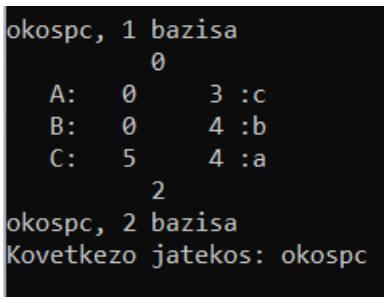
az első játékos következik

most már nem tud újra ő következni

bármi lesz a két lépés az ő bázisában 2 lesz a végén,

az ellenfél bázisában pedig 1

→ ezért a legelső helyes indexet választja, ami a B



3. lépéshez

a második játékos következik, ő csak 1 lépést tud előre nézni,

melyikkel szerez a legtöbbet

→ mindegyikkel ugyanaz lesz, ezért az a-t választja

(ami a tömbben 4. index)

```
okospc, 1 bazisa
         1
   A:    1     4 :c
   B:    0     5 :b
   C:    5     0 :a
         2
okospc, 2 bazisa
Kovetkezo jatekos: okospc
```

4. lépéshez

az első játékos következik

ha az első cellát választja akkor a játékszabály miatt 6-tal nő a bázisának értéke.

Mivel az utolsó cella saját és vele szemben olyan van, ami nem üres, ezért megkapja mindkettő golyóit

tehát 1-et és még, ami azzal szemben (5-ös index) van: 5 golyó

bármit is választ az ellenfél, ez a lehető legjobb lépés

→ A (2-es nehézséggel nézve biztosan ez)

```
okospc, 1 bazisa
         1
   A:    0     4 :c
   B:    0     0 :b
   C:    5     0 :a
         8
okospc, 2 bazisa
Kovetkezo jatekos: okospc
```

5. lépéshez

következik a második játékos, egyetlen cellát választhat

→ c

```
okospc, 1 bazisa
         2
   A:    1     0 :c
   B:    1     0 :b
   C:    6     0 :a
         8
okospc, 2 bazisa
Kovetkezo jatekos: okospc
```

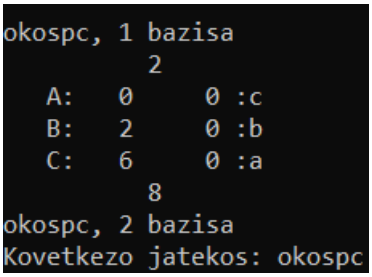
6. lépéshez

következik az első játékos

ha az első kettőből választ vége a játéknak és a lehető legtöbb golyóval nyer

ha az utolsót választja az első lépés után nem nő a különbség, a második után meg csak csökken

→ A-t fogja választani



A játék véget ért. Lehetne ellenőrizni is meg a maradékot pakolni, de arra van másik teszt.

4.6 Memóriakezelés tesztje

A memóriakezelést a memtrace modullal végeztem, melyet a https://git.ik.bme.hu/Prog2/ell_feladat/Test oldalról töltöttem le. A modul működésének érdekében a memtrace.h fájlt minden fájlban include-oltam. A modul nem jelzett memóriaszivárgást.