NHF - Gyarmathy Gábor

Generated by Doxygen 1.10.0

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 Army Class Reference

Sereg class és annak a függvényei.

`#include <Army.h>`

Public Member Functions

- **Army** ()

  Sereg default konstruktora. A méretét 0-ra állítja és egy üres-sztringet állít be a nevére.
- ∼**Army** ()

  Army objektum destuktora. Ebben felszabadul a bábuk listája amiben dinamikusan vannak tárolva a bábuk.
- const char ∗ getnameofArmy () const

  Getter függvény a sereg nevére.
- void setnameofArmy (const char ∗name)

  Setter függvény a sereg nevére.
- int getsizeofArmy () const

  Getter függvény a sereg méretére.
- void setsizeofArmy (int size)

  Setter függvény a sereg méretére.
- void **incrementsizeofArmy** ()

  Sereg méretének megnövelése 1-el.
- void addPiece (Piece &newPiece)

  Bábu hozzáadása a sereghez. Ha az adott bábunak túl nagyok az indexei, akkor nem adódik hozzá a sereghez.
- void deletePiece (int coordX, int coordY)

  Bábu eltávolítása a seregből. Koordináta alapján megadott bábu kitörlése. Ha nincs az adott mezőn bábu akkor nem töröl ki semmit.
- Piece ∗ getPiece (int coordX, int coordY)

  Getter függvény egy Piece-re mutató pointerért adott koordináták alapján. Keres a seregben az adott koordinátákon egy bábut és visszatér egy rámutató pointerrel. Amennyiben nincs ilyen bábu nullptr-el tér vissza.
- bool partOfArmy (Piece ∗piece)

  Logikai érték arra, hogy egy adott bábu része-e egy seregnek. Ezt leginkább a szimulációban használják a csapatok, hogy megállapítsák, hogy egy adott bábu szövetséges-e.
- Army (const Army &army)

  Sereg másolókonstruktor.
- Army & operator= (const Army &army)

  Sereg másolóoperátor.
- Piece ∗ getPiece (size_t idx)

  Getter függvény csapaton belüli i-edik bábunak. Leginkább arra van, hogy ciklusban végigmenjünk a csapaton és minden bábuhoz hozzáférhessünk.

Static Public Member Functions

- static void copyArmy (Army *source, Army *destination)

  Két paraméterként kapott sereg-pointerek között egyikből átmásol a másikba.
- static void mirrorArmy (Army *army)

  Áttükrözi egy seregnek a bábuit a másik térfélre. Szerkesztőben csak egy darab térfelet szerkesztünk, viszont a játék 2 összeillesztett térfél, ahol az egyik csapat fel van tükrözve a tábla tetejére. Az algoritmus nem tesz többet, mint a sereg összes bábujának az Y koordinátáját kivonja 9-ből. Emellett kisbetűssé teszi a bábuk betűjelét, ezzel jelezve, hogy a fekete csapatba tartoznak.

4.1.1 Detailed Description

Sereg class és annak a függvényei.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Army()

```
Army::Army (
            const Army & army )
```

Sereg másolókonstruktor.

Parameters

army	Lemásolandó sereg

4.1.3 Member Function Documentation

4.1.3.1 addPiece()

```
void Army::addPiece (
            Piece & newPiece )
```

Bábu hozzáadása a sereghez. Ha az adott bábunak túl nagyok az indexei, akkor nem adódik hozzá a sereghez.

Parameters

newPiece	Új bábu

4.1.3.2 copyArmy()

```
void Army::copyArmy (
            Army * source,
            Army * destination )  [static]
```

Két paraméterként kapott sereg-pointerek között egyikből átmásol a másikba.

Parameters

source	
destination	

4.1.3.3 deletePiece()

```
void Army::deletePiece (
            int coordX,
            int coordY )
```

Bábu eltávolítása a seregből. Koordináta alapján megadott bábu kitörlése. Ha nincs az adott mezőn bábu akkor nem töröl ki semmit.

Parameters

coordX	Eltávolítandó bábu x koordinátája.
coordY	Eltávolítandó bábu y koordinátája.

4.1.3.4 getnameofArmy()

```
const char * Army::getnameofArmy ( ) const
```

Getter függvény a sereg nevére.

Returns

A sereg neve.

4.1.3.5 getPiece() [1/2]

```
Piece * Army::getPiece (
            int coordX,
            int coordY )
```

Getter függvény egy Piece-re mutató pointerért adott koordináták alapján. Keres a seregben az adott koordinátákon egy bábut és visszatér egy rámutató pointerrel. Amennyiben nincs ilyen bábu nullptr-el tér vissza.

Parameters

coordX	Keresett bábu x koordinátája.
coordY	Keresett bábu y koordinátája.

Returns

Piece∗ Keresett bábura mutató pointer.

4.1.3.6 getPiece() [2/2]

```
Piece * Army::getPiece (
            size_t idx )
```

Getter függvény csapaton belüli i-edik bábunak. Leginkább arra van, hogy ciklusban végigmenjünk a csapaton és minden bábuhoz hozzáférhessünk.

Parameters

idx	Bábu indexe a listában.

Returns

Bábura mutató pointer.

4.1.3.7 getsizeofArmy()

```
int Army::getsizeofArmy ( ) const
```

Getter függvény a sereg méretére.

Returns

A sereg mérete.

4.1.3.8 mirrorArmy()

```
void Army::mirrorArmy (
            Army * army )  [static]
```

Áttükrözi egy seregnek a bábuit a másik térfélre. Szerkesztőben csak egy darab térfelet szerkesztünk, viszont a játék 2 összeillesztett térfél, ahol az egyik csapat fel van tükrözve a tábla tetejére. Az algoritmus nem tesz többet, mint a sereg összes bábujának az Y koordinátáját kivonja 9-ből. Emellett kisbetűssé teszi a bábuk betűjelét, ezzel jelezve, hogy a fekete csapatba tartoznak.

Parameters

army	A feltükrözendő sereg.

4.1.3.9 operator=()

```
Army & Army::operator= (
            const Army & army )
```

Sereg másolóoperátor.

Parameters

army	Lemásolandó sereg.

Returns

Army& Lemásolt sereg referenciája.

4.1.3.10 partOfArmy()

```
bool Army::partOfArmy (
            Piece * piece )
```

Logikai érték arra, hogy egy adott bábu része-e egy seregnek. Ezt leginkább a szimulációban használják a csapatok, hogy megállapítsák, hogy egy adott bábu szövetséges-e.

Parameters

piece	Bábura mutató pointer

Returns

true: Része a seregnek

false: Nem része a seregnek

4.1.3.11 setnameofArmy()

```
void Army::setnameofArmy (
            const char * name )
```

Setter függvény a sereg nevére.

Parameters

name	Beállítandó név.

4.1.3.12 setsizeofArmy()

```
void Army::setsizeofArmy (
            int size )
```

Setter függvény a sereg méretére.

Parameters

size	Beállítandó méret.

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

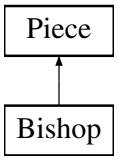
- Army.h
- Army.cpp

4.2 Bishop Class Reference

Bishop osztály, ami a futót reprezentálja a sakkjátékban.

`#include <Bishop.h>`

Inheritance diagram for Bishop:



Public Member Functions

- Bishop (int x, int y)

  Bishop osztály konstruktora.
- void calculateMoves (Game *game) override

  Lehetséges lépéseket számolja ki a futónak. A lépések kiszámolására az elkészített diagonal() -átlós lépések függvényét használja.

Public Member Functions inherited from Piece

- Piece (char name, int coordX=0, int coordY=0)

  Bábu konstruktor.
- virtual ~**Piece** ()

  Bábu destruktor. Amikor lefut akkor felszabadul a listája ami tárolja a lehetséges lépéseit(Persze, csak ha játékban van, mivel csak akkor vannak számolva a lépései).
- int getCoordY () const

  Getter függvény az y koordinátára.
- int getCoordX () const

  Getter függvény az x koordinátára.
- char getName () const

  Getter függvény a bábu nevére.
- List< Move > & getMoves ()

  Getter függvény a bábu lépéseinek a listájára.
- void setCoordY (int newY)

  Setter függvény a bábu Y koordinátájának.
- void setCoordX (int newX)

  Setter függvény a bábu x koordinátájának.
- bool operator== (const Piece &otherPiece) const

  Egyenlőség operátor.
- void **toLowercase** ()

  Átváltja egy bábunak a nevét kisbetűsre nagybetűsről. Ezt a tükrözésnél használja a program.
- void addMove (char destinationPieceName, int coordX, int coordY)

  Hozzáad egy dinamikusan foglalt Move objektumot a meglévő Piecenek a Move listájához. Ezt a függvényt majd a checkAndAddMove(...) használja majd.

Additional Inherited Members

Static Public Member Functions inherited from Piece

- static Piece * createPiece (char name, int x, int y)

  Létrehoz egy Piece objektumot. Név alapján hozza létre a bábut. Ha rossz nevet kap, akkor nullptr-t ad vissza.
- static bool checkAndAddMove (Game *game, Piece *originPiece, int posX, int posY)

  Kiszámolja, hogy adott mezőre léphet-e a Bábu. Ha igen akkor hozzáadja a lépéseihez. Emellett visszatér egy bool value-val, mivel a legtöbb bábu egy hosszabb vonalban lép.
- static void upwards (Piece *originPiece, Game *game)

  Felfelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void downwards (Piece *originPiece, Game *game)

  Lefelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void rightwards (Piece *originPiece, Game *game)

  Jobbrafelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void leftwards (Piece *originPiece, Game *game)

  Balrafelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void diagonalUpRight (Piece *originPiece, Game *game)

  Északkelet felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void diagonalUpLeft (Piece *originPiece, Game *game)

  Északnyugat felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void diagonalDownRight (Piece *originPiece, Game *game)

  Délkelet felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void diagonalDownLeft (Piece *originPiece, Game *game)

  Délnyugat felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void orthogonal (Piece *originPiece, Game *game)

  Ortogonális lépések(+) Így lép például a bástya. Ez pedig az upwards(),downwards(),rightwards(),leftwards() függvények összesítése.
- static void diagonal (Piece *originPiece, Game *game)

  Átlós lépések(X) Így lép például a futó. Ez pedig a diagonalDownLeft(),diagonalDownRight(),diagonalUpLeft(),diagonalUpRight() függvények összesítése.
- static void pawnMove (Piece *originPiece, Game *game)

  Parasztlépés. Algoritmus leírása:
- static void horseMove (Piece *originPiece, Game *game)

  Lólépés. Ló bábú lépéseinek ellenőrzése és hozzáadása a lehetséges lépésekhez.
- static void kingMove (Piece *originPiece, Game *game)

  Királylépés. ellenőrzi a megadott király bábú lehetséges lépéseit a táblán, majd hozzáadja azokat a lehetséges lépések listájához.

Protected Attributes inherited from Piece

- List< Move > piece_moves

4.2.1 Detailed Description

Bishop osztály, ami a futót reprezentálja a sakkjátékban.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Bishop()

```
Bishop::Bishop (
            int x,
            int y )
```

Bishop osztály konstruktora.

Parameters

x	Az oszlop koordinátája
y	A sor koordinátája

4.2.3 Member Function Documentation

4.2.3.1 calculateMoves()

```
void Bishop::calculateMoves (
            Game * game )  [override], [virtual]
```

Lehetséges lépéseket számolja ki a futónak. A lépések kiszámolására az elkészített diagonal() -átlós lépések függvényét használja.

Parameters

game	A játék objektumra mutató pointer

Implements Piece.

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

- Bishop.h
- Bishop.cpp

4.3 Button Class Reference

A gomb class. Ezekből épülnek fel a menük. Minden gombnak van egy id-je. Ezeket az id-ket a gombot tartalmazó Menü fogja kiosztani a gomboknak amilyen sorrendben jönnek. Minden gomb tartalmaz egy funkciót ami futtathat menüt/szerkesztőt/játékot.

```
#include <button.h>
```

Public Member Functions

- **Button** ()

  Alapértelmezett konstruktor.
- Button (const char ∗name, size_t id, void(∗function)())

  Paraméterezett konstruktor egyetlen függvénnyel.
- Button (const char ∗name, size_t id, void(∗functionArmy)(Army ∗), Army ∗armyPtr)

  Paraméterezett konstruktor egy függvénnyel, és egy Army objektummal.
- Button (const char ∗name, size_t id, void(∗functionArmy)(Army ∗first, Army ∗second), Army ∗armyPtr1, Army ∗armyPtr2)

  Paraméterezett konstruktor két függvénnyel és két Army objektummal.
- ∼**Button** ()

  Destruktor. Felszabadítja a dinamikusan foglalt gomb nevet.
- **Button** (const Button &other)

  Másoló konstruktor.
- Button & **operator=** (const Button &other)

  Értékadó operátor.
- void setId (size_t id)

  Setter függvény az id-ra.
- size_t getId () const

  Getter függvény az id-ra.
- const char ∗ getName () const

  Getter függvény a névre.
- ButtonFunctionHandler getFunction () const

  Gombhoz rendelt funkció lekérése.

4.3.1 Detailed Description

A gomb class. Ezekből épülnek fel a menük. Minden gombnak van egy id-je. Ezeket az id-ket a gombot tartalmazó Menü fogja kiosztani a gomboknak amilyen sorrendben jönnek. Minden gomb tartalmaz egy funkciót ami futtathat menüt/szerkesztőt/játékot.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Button() [1/3]

```
Button::Button (
            const char * name,
            size_t id,
            void(*)() function )
```

Paraméterezett konstruktor egyetlen függvénnyel.

Parameters

name	A gomb neve
id	A gomb sorszáma
function	A gombhoz rendelt függvény pointer

4.3.2.2 Button() [2/3]

```
Button::Button (
            const char * name,
            size_t id,
            void(*)(Army *) functionArmy,
            Army * armyPtr )
```

Paraméterezett konstruktor egy függvénnyel, és egy Army objektummal.

Parameters

name	A gomb neve
id	A gomb sorszáma
functionArmy	A gombhoz rendelt függvény pointer egy Army objektummal
armyPtr	Az Army objektum pointer

4.3.2.3 Button() [3/3]

```
Button::Button (
            const char * name,
            size_t id,
            void(*)(Army *first, Army *second) functionArmy,
            Army * armyPtr1,
            Army * armyPtr2 )
```

Paraméterezett konstruktor két függvénnyel és két Army objektummal.

Parameters

name	A gomb neve
id	A gomb sorszáma
functionArmy	A gombhoz rendelt függvény pointer két Army objektummal
armyPtr1	Az első Army objektum pointer
armyPtr2	A második Army objektum pointer

4.3.3 Member Function Documentation

4.3.3.1 getFunction()

```
ButtonFunctionHandler Button::getFunction ( ) const
```

Gombhoz rendelt funkció lekérése.

Returns

A gombhoz rendelt funkció

4.3.3.2 getId()

```
size_t Button::getId ( ) const
```

Getter függvény az id-ra.

Returns

A gomb sorszáma

4.3.3.3 getName()

```
const char * Button::getName ( ) const
```

Getter függvény a névre.

Returns

A gomb neve

4.3.3.4 setId()

```
void Button::setId (
            size_t id )
```

Setter függvény az id-ra.

Parameters

id	Az új id

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

- button.h
- button.cpp

4.4 ButtonFunctionHandler Class Reference

Hozzákapcsolja a function-pointereket a gombokhoz. Emellett lehetővé teszi, hogy több fajta függvényeket is tudjon futtatni egy gomb, akár olyanokat is amiknek más paramétereik vannak.

```
#include <ButtonFunctionHandler.h>
```

Public Member Functions

- ~**ButtonFunctionHandler** ()

  Destruktor.
- **ButtonFunctionHandler** ()

  Az osztály alapértelmezett konstruktora.
- ButtonFunctionHandler (void(*func)())

  Konstruktor, amely egy függvényt kap paraméterül.
- ButtonFunctionHandler (void(*funcArmy)(Army *), Army *armyPtr)

  Konstruktor, amely egy függvényt és egy Army objektum pointert kap paraméterül.
- ButtonFunctionHandler (void(*funcArmy)(Army *, Army *), Army *armyPtr1, Army *armyPtr2)

  Konstruktor, amely két függvényt és két Army objektum pointert kap paraméterül.
- void **execute** ()

  Az objektum megfelelő függvényének végrehajtása.

4.4.1 Detailed Description

Hozzákapcsolja a function-pointereket a gombokhoz. Emellett lehetővé teszi, hogy több fajta függvényeket is tudjon futtatni egy gomb, akár olyanokat is amiknek más paramétereik vannak.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 ButtonFunctionHandler() [1/3]

```
ButtonFunctionHandler::ButtonFunctionHandler (
            void(*)() func )  [inline]
```

Konstruktor, amely egy függvényt kap paraméterül.

Parameters

func	Paraméternélküli függvénypointer. Ehhez nem kellenek seregek.

4.4.2.2 ButtonFunctionHandler() [2/3]

```
ButtonFunctionHandler::ButtonFunctionHandler (
            void(*)(Army *) funcArmy,
            Army * armyPtr )  [inline]
```

Konstruktor, amely egy függvényt és egy Army objektum pointert kap paraméterül.

Parameters

funcArmy	Army paraméterrel rendelkező függvénypointer. Ennek már szüksége van egy seregre, hogy működjön.
armyPtr	Az Army objektum pointer.

4.4.2.3 ButtonFunctionHandler() [3/3]

```
ButtonFunctionHandler::ButtonFunctionHandler (
            void(*)(Army *, Army *) funcArmy,
            Army * armyPtr1,
            Army * armyPtr2 )  [inline]
```

Konstruktor, amely két függvényt és két Army objektum pointert kap paraméterül.

Parameters

funcArmy	Két Army paraméterrel rendelkező függvénypointer. Ez olyan függvényeket futtat gombnyomásra, ahol 2 sereg is kell.
armyPtr1	Az első Army objektum pointer.
armyPtr2	A második Army objektum pointer.

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

- ButtonFunctionHandler.h
- ButtonFunctionHandler.cpp

4.5 ButtonFunctions Class Reference

Függvénypointerek, amik aktiválódnak ha lenyomják az őket tartalmazó gombot.

```
#include <buttonfunctions.h>
```

Static Public Member Functions

- static void MainMenu ()
- static void Play ()

  Ebben a menüben kell választani aközött, hogy szerkeszteni vagy játszani szeretnénk Működése:
- static void NewGame ()

  Menü a játék futtatása előtt.
- static void **ArmyMenu** ()

  Létrehoz egy menüt ami kimutatja az összes sereget ami le van mentve. Használt ez arra, hogy az editornál kiválasszunk egy sereget editelésre.
- static void **CreateArmy** ()

  Megnyit egy editor-t paraméternélküli konstruktorral, ezzel új sereget lehet létrehozni.
- static void **EditArmy** (Army *)

  Sereg szerkesztésére használt fv.
- static void **ChooseArmy** (Army *)

  Sereg kiválasztására használt fv. A sereg amit kiválaszt az armies.txt-ből kiválasztott listából, azt tölti majd át a paraméterként kapott army-be.*
- static void PlayMatch (Army *reg1, Army *reg2)

  Ez indítja el a játékot.

4.5.1 Detailed Description

Függvénypointerek, amik aktiválódnak ha lenyomják az őket tartalmazó gombot.

4.5.2 Member Function Documentation

4.5.2.1 MainMenu()

```
void ButtonFunctions::MainMenu ( )  [static]
```

Főmenü függvény

4.5.2.2 NewGame()

```
void ButtonFunctions::NewGame ( )  [static]
```

Menü a játék futtatása előtt.

Van benne 2 db army amikbe áttudja tölteni a felhasználó a fájlból a kiválasztott seregeit. Majd azokat a seregeket adja át paraméterként a játék futtatásánál.

4.5.2.3 Play()

```
void ButtonFunctions::Play ( )  [static]
```

Ebben a menüben kell választani aközött, hogy szerkeszteni vagy játszani szeretnénk Működése:

- Létrejön egy Menu objektum.
- Hozzáadódnak a szükséges gombok, a neveikkel funkcióikkal.
- Futtatja a Menüt->választ a felhaszáló a gombok közül.

4.5.2.4 PlayMatch()

```
void ButtonFunctions::PlayMatch (
            Army * reg1,
            Army * reg2 )  [static]
```

Ez indítja el a játékot.

Parameters

reg1	Fehér csapat
reg2	Fekete csapat

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

- buttonfunctions.h
- buttonfunctions.cpp

4.6 Computer Class Reference

Computer class az, aki kiszámolja a lehetséges lépéseket. Megállapítja, hogy melyik csapat jön a játékban, majd kigyűjti a csapat lépéseit.

`#include <Computer.h>`

Public Member Functions

- **Computer** ()

  Computer konstruktor.
- void calculateMoves (Game *game)

  A játékban következőként lépő csapat összes lehetséges lépését kiszámolja. Algoritmus leírása:
- Move * decideMove (Game *game)

  A csapatnak az összesített lépés-listájából kiválasztja a legnagyobb súlyú lépést. Ha a legnagyobb súlyú lépés értéke 0, akkor random választ az összes lépésből. Ha pedig több legnagyobb van, akkor azokból az legelső megtalálttal tér vissza.

4.6.1 Detailed Description

Computer class az, aki kiszámolja a lehetséges lépéseket. Megállapítja, hogy melyik csapat jön a játékban, majd kigyűjti a csapat lépéseit.

4.6.2 Member Function Documentation

4.6.2.1 calculateMoves()

```
void Computer::calculateMoves (
            Game * game )
```

A játékban következőként lépő csapat összes lehetséges lépését kiszámolja. Algoritmus leírása:

- Megállapítja melyik csapat jön
- Majd annak a csapatnak az összes bábujának kiszámolja a lehetséges lépéseit, amiket minden bábu külön tárol egy saját listában.
- Ezeket a lépéseket belerakja a csapatnak a saját összesített lépés-listájába.

Parameters

game	Az éppen futó játék.

4.6.2.2 decideMove()

```
Move * Computer::decideMove (
            Game * game )
```

A csapatnak az összesített lépés-listájából kiválasztja a legnagyobb súlyú lépést. Ha a legnagyobb súlyú lépés értéke 0, akkor random választ az összes lépésből. Ha pedig több legnagyobb van, akkor azokból az legelső megtalálttal tér vissza.

Parameters

game	Az éppen futó játék.

Returns

A lépés amit a csapat fog lépni a játékban.

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

- Computer.h
- Computer.cpp

4.7 Editor Class Reference

A seregek létrehozásáért és szerkesztéséért felelős osztály.

`#include <Editor.h>`

Public Member Functions

- **Editor** ()
  Default konstruktor. Ha a felhasználó új sereget szeretne létrehozni.
- ~**Editor** ()
  Destruktor. Felszabadítja a dinamikusan foglalt sereget.
- Editor (Army *army)
  Paraméteres konstruktor. Egy meglévő sereget szerkeszt.
- Army * getArmy () const
  Getter függvény az éppen szerkesztett sereg pointerére.
- Piece * searchFor (int coordX, int coordY)
  Keres egy bábut a megadott x-y koordinátákon.
- void **updateExit** ()
  Frissíti a kilépési boolt. Ha hamis volt igazzá teszi.
- bool getExit () const
  Getter függvény a kilépési boolra.
- void **updateDelete** ()
  Elrendelteti az éppen szerkesztett seregnek a törlését a lementett seregek közül.
- bool getDelete () const
  Getter függvény a törlési bool-ra.
- void **saveArmy** ()
  Elmenti az újonnan készült sereget. Kimenti az armies.txt-be leghátulra.
- void **editArmy** ()
  Elmenti az éppen szerkesztett sereg változtatásait. Átírja az armies.txt-ben a sereget a legújabb verziójával.
- void **deleteArmy** ()
  Kitörli az éppen szerkesztett sereget. Kitörli az armies.txt-ben a sereget.

4.7.1 Detailed Description

A seregek létrehozásáért és szerkesztéséért felelős osztály.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 Editor()

```
Editor::Editor (
            Army * army )
```

Paraméteres konstruktor. Egy meglévő sereget szerkeszt.

Parameters

army	Pointer arra a dinamikusan foglalt seregre amit szerkesztünk.

4.7.3 Member Function Documentation

4.7.3.1 getArmy()

```
Army * Editor::getArmy ( ) const
```

Getter függvény az éppen szerkesztett sereg pointerére.

Returns

Az éppen szerkesztett sereg pointere.

4.7.3.2 getDelete()

```
bool Editor::getDelete ( ) const
```

Getter függvény a törlési bool-ra.

Returns

Igaz, ha a szerkesztett sereget törölni kell, egyébként hamis.

4.7.3.3 getExit()

```
bool Editor::getExit ( ) const
```

Getter függvény a kilépési boolra.

Returns

Igaz, ha a szerkesztőből ki kell lépni, egyébként hamis.

4.7.3.4 searchFor()

```
Piece * Editor::searchFor (
            int coordX,
            int coordY )
```

Keres egy bábut a megadott x-y koordinátákon.

Parameters

coordX	A keresett mező x koordinátája.
coordY	A keresett mező y koordinátája.

Returns

Pointer a keresett bábura. Ha nincsen bábu a keresett mezőn akkor nullptr a visszatérési érték.

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

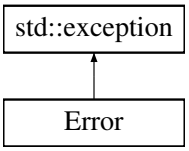
- Editor.h
- Editor.cpp

4.8 Error Class Reference

Saját error osztály. Az osztály az std::exception-ből származik és külön hibaüzenetet produkál.

`#include <Error.h>`

Inheritance diagram for Error:



Public Member Functions

- Error (std::string &message)

  Error objektumnak a konstruktora adott hibaüzenettel.
- const char * what () const noexcept

  Visszatér egy C típusú sztringgel, ami leírja a hiba típusát.

4.8.1 Detailed Description

Saját error osztály. Az osztály az std::exception-ből származik és külön hibaüzenetet produkál.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 Error()

```
Error::Error (
            std::string & message )
```

Error objektumnak a konstruktora adott hibaüzenettel.

Parameters

message	Hibaüzenet.

4.8.3 Member Function Documentation

4.8.3.1 what()

```
const char * Error::what ( ) const  [noexcept]
```

Visszatér egy C típusú sztringgel, ami leírja a hiba típusát.

Returns

Hibaüzenet.

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

- Error.h
- Error.cpp

4.9 Filemanagement Class Reference

Fájlkezelő osztály. Legfőkébb az elkészített seregek tárolását valósítja meg az armies.txt-ben.

`#include <Filemanagement.h>`

Static Public Member Functions

- static List< Army > ListofArmies (const char *filename)

  A lementett seregek listájával visszatérő fv. Kiolvassa az adott fájból(armies.txt) a seregeket. Ha nem létezik akkor létrehoz egyet 0 db sereggel.
- static void AppendArmy (Army *army, const char *filename)

  Sereg hozzáfűzése a fájlhoz. A fájl legvégéhez odafűzi a paraméterként kapott sereget.
- static void EditArmy (Army *army, const char *filename)

  Sereg átszerkesztése fájlon belül. Algoritmus leírása:
- static void DeleteArmy (Army *army, const char *filename)

4.9.1 Detailed Description

Fájlkezelő osztály. Legfőkébb az elkészített seregek tárolását valósítja meg az armies.txt-ben.

4.9.2 Member Function Documentation

4.9.2.1 AppendArmy()

```
void Filemanagement::AppendArmy (
            Army * army,
            const char * filename )  [static]
```

Sereg hozzáfűzése a fájlhoz. A fájl legvégéhez odafűzi a paraméterként kapott sereget.

Parameters

army	A hozzáfűzendő seregre mutató pointer
filename	A megnyitandó fájl neve

4.9.2.2 DeleteArmy()

```
void Filemanagement::DeleteArmy (
            Army * army,
            const char * filename )  [static]
```

- A overwriteline függvényt használva az első sort írjuk felül: 1-el csökkentjük a sorszámát.
- Ismételten létrehozunk lines1 és lines2 vektorokat.
- lines1 tartalmazza azokat a sorokat, amelyeket nem kell módosítanunk.
- Végigiterálunk lines1-en a kitörlendő névig.
- Kihagyjuk az Army-t.
- lines2-be olvassuk a fennmaradó sorokat.
- Kiírjuk lines1 és lines2 tartalmát

Parameters

army	Kitörlendő sereg
filename	Fájlnév

4.9.2.3 EditArmy()

```
void Filemanagement::EditArmy (
            Army * army,
            const char * filename )  [static]
```

Sereg átszerkesztése fájlon belül. Algoritmus leírása:

- Beolvassuk a sorokat.
- Név alapján kikeressük az Army-t, amit editelünk.
- Kiolvassuk, hogy hány bábu van az Army-ben.
- Ebből tudjuk, hogy hány sort kell majd ignorálni.
- A beolvasás második részét egy 2. stringvektorba tároljuk.
- Bezárjuk a beolvasást.
- Írásra megnyitjuk a fájlt.
- Beleírjuk az első stringvektor tartalmát, ami az editelendő Army előtt áll.
- Az Army∗ army alapján beleírjuk a fájlba az Army-t.
- Beleírjuk a második stringvektort a fájlba.

Parameters

army	A szerkesztendő sereg
filename	Fájlnév

4.9.2.4 ListofArmies()

```
List< Army > Filemanagement::ListofArmies (
            const char * filename )  [static]
```

A lementett seregek listájával visszatérő fv. Kiolvassa az adott fájból(armies.txt) a seregeket. Ha nem létezik akkor létrehoz egyet 0 db sereggel.

Parameters

filename	Az olvasandó fájl neve

Returns

List<Army> A lementett seregek listája

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

- Filemanagement.h
- Filemanagement.cpp

4.10 Game Class Reference

A játék class. Ez van használatban amikor 2 csapat játszik.

`#include <Game.h>`

Public Member Functions

- **Game** ()

  Game default konstruktor.
- ~**Game** ()

  Game destruktor Felszabadítja a 2 dinamikusan lefoglalt csapatot a játékban.
- Game (Army *white, Army *black)

  Game konstruktor 2 paraméterként átadott sereggel. Lemásolja magának a seregeket a 2 Team objektumon belül.
- bool isWhiteTurn () const

  Getter függvény a WhiteTurn bool-ra. Bool érték annak az eldöntésére, hogy melyik csapat jön.
- bool getEnd () const

  Getter függvény a játék-végét eldöntő bool-ra. Ha igaz, akkor nem lesz több lépés számolva, vége van a játéknak.
- GameResult getResult () const

  Getter függvény a végeredményre.

- void **updateEnd** ()

  Átállítja az EndOfGame boolt az ellentetjére. A kódban arra van használva, hogy véget vessen a játéknak.
- Piece * searchFor (int x, int y)

  Megnézi, hogy van-e bábu az adott mezőn(x,y) Ha megtalálja akkor visszatér egy bábupointerrel. Ha nincs semmi a mezőn akkor nullpointerrel.
- Team * getTeam (size_t idx)

  Getter csapat objektumra index alapján 0-ás index idx fehér csapat, 1-es index idx fekete.
- TeamColor getColorOfPiece (Piece *piece)

  Visszaadja egy paraméterként átadott bábunak a színét a játéban.
- bool occupied (int x, int y)

  Visszaadja, hogy van-e bármelyik csapatból is bábu az adott mezőn. Mindkét csapaton átmegy és ha mindkettő nullptr-t ad vissza a getPiece(x,y)-ból, akkor a mezőn nincs egy bábu sem.
- void makeMove ()

  Amikor lefut, akkor a játék megtesz egy lépést azzal a csapattal akinek a köre van. Algoritmus leírása:
- void **playRound** ()

  Lejátszik egy darab kört a játékból. Ha vége lenne a játéknak, akkor egyből véget vet neki mielőtt megtenne egy lépést is. Ha meg nem lenne vége a játéknak, kiszámolja a lépést, megteszi, majd átadja a kört, azzal, hogy megváltoztatja a WhiteTurn bool-t az ellentetjére.
- void checkIfOver ()

  Megállapítja, hogy vége van-e a játéknak. Ha vége van a játéknak, akkor frissíti az endOfGame boolt, emellett eldönti, hogy mi a játék végeredménye és beállítja. A játék végeredményének eldöntésének módja:
- void **clearMovesBuffer** ()

  Kitisztítja a csapat összesített lépés-listáját. Annak a csapatnak szabadítja fel ezt a listáját akinek a köre most van(Természetesen a függvény ami ezt használja ezután adja át a kört).

4.10.1 Detailed Description

A játék class. Ez van használatban amikor 2 csapat játszik.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 Game()

```
Game::Game (
            Army * white,
            Army * black )
```

Game konstruktor 2 paraméterként átadott sereggel. Lemásolja magának a seregeket a 2 Team objektumon belül.

Parameters

white	Fehér csapat. Ez a csapat lesz a 0.-ás indexű csapat a team tömbben.
black	Fekete csapat. Ez a csapat lesz az 1.-es indexű csapat a team tömbben. Ez a csapat még fel is lesz tükrözve a tábla túloldalára.

4.10.3 Member Function Documentation

4.10.3.1 checkIfOver()

```
void Game::checkIfOver ( )
```

Megállapítja, hogy vége van-e a játéknak. Ha vége van a játéknak, akkor frissíti az endOfGame boolt, emellett eldönti, hogy mi a játék végeredménye és beállítja. A játék végeredményének eldöntésének módja:

- Ha a jelenlegi körben következő csapatnak 0 legális lépése van, akkor automatikusan döntetlen lesz.
- Ha a jelenlegi körben következő csapatnak nincsen királya a táblán, akkor automatikusan a másik csapat nyer.
- Egyébként meg folytatódik a játék

4.10.3.2 getColorOfPiece()

```
TeamColor Game::getColorOfPiece (
            Piece * piece )
```

Visszaadja egy paraméterként átadott bábunak a színét a játéban.

Parameters

piece	Bábu aminek keressük a színét.

Returns

A kérdéses bábunak a csapatszíne.

4.10.3.3 getEnd()

```
bool Game::getEnd ( ) const
```

Getter függvény a játék-végét eldöntő bool-ra. Ha igaz, akkor nem lesz több lépés számolva, vége van a játéknak.

Returns

true: Vége a játéknak.

false: Folytatódik a játék.

4.10.3.4 getResult()

```
GameResult Game::getResult ( ) const
```

Getter függvény a végeredményre.

Returns

A játék végeredménye

4.10.3.5 getTeam()

```
Team * Game::getTeam (
            size_t idx )
```

Getter csapat objektumra index alapján 0-ás index idx fehér csapat, 1-es index idx fekete.

Parameters

idx	Keresett csapat indexe.

Returns

Keresett csapatra mutató pointer.

4.10.3.6 isWhiteTurn()

```
bool Game::isWhiteTurn ( ) const
```

Getter függvény a WhiteTurn bool-ra. Bool érték annak az eldöntésére, hogy melyik csapat jön.

Returns

true: Fehér csapat jön

false: Fekete csapat jön

4.10.3.7 makeMove()

```
void Game::makeMove ( )
```

Amikor lefut, akkor a játék megtesz egy lépést azzal a csapattal akinek a köre van. Algoritmus leírása:

- Kiszámolja a csapat lépését a Computer objektum és rámutat a kiválasztott lépésre egy Move pointerrel
- A lépésből megkapja, hogy kivel kell lépnie, amiből pedig a csapat színét is.
- Megnézi, hogy a mező amire lép el van-e foglalva ellenséges bábuval, és ha igen akkor eltávolítja azt az ellenséges bábut
- Majd átállítja az új koordinátákra a bábunak a helyét.
- Felszabadítja az előbb jött csapatnak az összesített lépés-listáját.

4.10.3.8 occupied()

```
bool Game::occupied (
            int x,
            int y )
```

Visszaadja, hogy van-e bármelyik csapatból is bábu az adott mezőn. Mindkét csapaton átmegy és ha mindkettő nullptr-t ad vissza a getPiece(x,y)-ból, akkor a mezőn nincs egy bábu sem.

Parameters

x	Mező x koordinátája.
y	Mező y koordinátája.

Returns

true: Van rajta bábu.

false: Nincs rajta bábu.

4.10.3.9 searchFor()

```
Piece * Game::searchFor (
            int x,
            int y )
```

Megnézi, hogy van-e bábu az adott mezőn(x,y) Ha megtalálja akkor visszatér egy bábupointerrel. Ha nincs semmi a mezőn akkor nullpointerrel.

Parameters

x	A keresett mező x koordinátája.
y	A keresett mező y koordinátája.

Returns

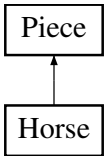
Adott mezőn álló bábura mutató pointer.

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

- Game.h
- Game.cpp

4.11 Horse Class Reference

Inheritance diagram for Horse:



Public Member Functions

- Horse (int x, int y)

  Horse osztály konstruktora.
- void calculateMoves (Game *game) override

  Lehetséges lépéseket számolja ki a lónak. Az erre külön elkészített horseMove()-t használja.

Public Member Functions inherited from Piece

- Piece (char name, int coordX=0, int coordY=0)

  Bábu konstruktor.
- virtual ~**Piece** ()

  Bábu destruktor. Amikor lefut akkor felszabadul a listája ami tárolja a lehetséges lépéseit(Persze, csak ha játékban van, mivel csak akkor vannak számolva a lépései).
- int getCoordY () const

  Getter függvény az y koordinátára.
- int getCoordX () const

  Getter függvény az x koordinátára.
- char getName () const

  Getter függvény a bábu nevére.
- List< Move > & getMoves ()

  Getter függvény a bábu lépéseinek a listájára.
- void setCoordY (int newY)

  Setter függvény a bábu Y koordinátájának.
- void setCoordX (int newX)

  Setter függvény a bábu x koordinátájának.
- bool operator== (const Piece &otherPiece) const

  Egyenlőség operátor.
- void **toLowercase** ()

  Átváltja egy bábunak a nevét kisbetűsre nagybetűsről. Ezt a tükrözésnél használja a program.
- void addMove (char destinationPieceName, int coordX, int coordY)

  Hozzáad egy dinamikusan foglalt Move objektumot a meglévő Piecenek a Move listájához. Ezt a függvényt majd a checkAndAddMove(...) használja majd.

Additional Inherited Members

Static Public Member Functions inherited from Piece

- static Piece * createPiece (char name, int x, int y)

  Létrehoz egy Piece objektumot. Név alapján hozza létre a bábut. Ha rossz nevet kap, akkor nullptr-t ad vissza.
- static bool checkAndAddMove (Game *game, Piece *originPiece, int posX, int posY)

  Kiszámolja, hogy adott mezőre léphet-e a Bábu. Ha igen akkor hozzáadja a lépéseihez. Emellett visszatér egy bool value-val, mivel a legtöbb bábu egy hosszabb vonalban lép.
- static void upwards (Piece *originPiece, Game *game)

  Felfelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void downwards (Piece *originPiece, Game *game)

  Lefelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void rightwards (Piece *originPiece, Game *game)

  Jobbrafelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void leftwards (Piece *originPiece, Game *game)

  Balrafelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void diagonalUpRight (Piece *originPiece, Game *game)

  Északkelet felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void diagonalUpLeft (Piece *originPiece, Game *game)

Északnyugat felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.

- static void diagonalDownRight (Piece *originPiece, Game *game)

  Délkelet felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void diagonalDownLeft (Piece *originPiece, Game *game)

  Délnyugat felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void orthogonal (Piece *originPiece, Game *game)

  Ortogonális lépések(+) Így lép például a bástya. Ez pedig az upwards(),downwards(),rightwards(),leftwards() függvények összesítése.
- static void diagonal (Piece *originPiece, Game *game)

  Átlós lépések(X) Így lép például a futó. Ez pedig a diagonalDownLeft(),diagonalDownRight(),diagonalUpLeft(),diagonalUpRight() függvények összesítése.
- static void pawnMove (Piece *originPiece, Game *game)

  Parasztlépés. Algoritmus leírása:
- static void horseMove (Piece *originPiece, Game *game)

  Lólépés. Ló bábú lépéseinek ellenőrzése és hozzáadása a lehetséges lépésekhez.
- static void kingMove (Piece *originPiece, Game *game)

  Királylépés. ellenőrzi a megadott király bábú lehetséges lépéseit a táblán, majd hozzáadja azokat a lehetséges lépések listájához.

Protected Attributes inherited from Piece

- List< Move > piece_moves

4.11.1 Constructor & Destructor Documentation

4.11.1.1 Horse()

```
Horse::Horse (
            int x,
            int y )
```

Horse osztály konstruktora.

Parameters

x	Az oszlop koordinátája
y	A sor koordinátája

4.11.2 Member Function Documentation

4.11.2.1 calculateMoves()

```
void Horse::calculateMoves (
            Game * game )  [override], [virtual]
```

Lehetséges lépéseket számolja ki a lónak. Az erre külön elkészített horseMove()-t használja.

Parameters

game	A játék objektumra mutató pointer

Implements Piece.

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

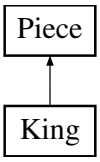
- Horse.h
- Horse.cpp

4.12 King Class Reference

King osztály, ami a királyt reprezentálja a sakkjátékban. Ebből a bábuból minden csapatnak kell, hogy legyen különben automatikusan veszt. Ez a játék feltétele. Ebből lehet több egy csapatban.

`#include <King.h>`

Inheritance diagram for King:



Public Member Functions

- King (int x, int y)

  King osztály konstruktora.
- void calculateMoves (Game *game) override

  Lehetséges lépéseket számolja ki a királynak. A külön erre elkészített kingMove() fv.-t használja.

Public Member Functions inherited from Piece

- Piece (char name, int coordX=0, int coordY=0)

  Bábu konstruktor.
- virtual ~**Piece** ()

  Bábu destruktor. Amikor lefut akkor felszabadul a listája ami tárolja a lehetséges lépéseit(Persze, csak ha játékban van, mivel csak akkor vannak számolva a lépései).
- int getCoordY () const

  Getter függvény az y koordinátára.
- int getCoordX () const

  Getter függvény az x koordinátára.
- char getName () const

  Getter függvény a bábu nevére.
- List< Move > & getMoves ()

  Getter függvény a bábu lépéseinek a listájára.

- void setCoordY (int newY)

  Setter függvény a bábu Y koordinátájának.
- void setCoordX (int newX)

  Setter függvény a bábu x koordinátájának.
- bool operator== (const Piece &otherPiece) const

  Egyenlőség operátor.
- void **toLowercase** ()

  Átváltja egy bábunak a nevét kisbetűsre nagybetűsről. Ezt a tükrözésnél használja a program.
- void addMove (char destinationPieceName, int coordX, int coordY)

  Hozzáad egy dinamikusan foglalt Move objektumot a meglévő Piecenek a Move listájához. Ezt a függvényt majd a checkAndAddMove(...) használja majd.

Additional Inherited Members

Static Public Member Functions inherited from Piece

- static Piece * createPiece (char name, int x, int y)

  Létrehoz egy Piece objektumot. Név alapján hozza létre a bábut. Ha rossz nevet kap, akkor nullptr-t ad vissza.
- static bool checkAndAddMove (Game *game, Piece *originPiece, int posX, int posY)

  Kiszámolja, hogy adott mezőre léphet-e a Bábu. Ha igen akkor hozzáadja a lépéseihez. Emellett visszatér egy bool value-val, mivel a legtöbb bábu egy hosszabb vonalban lép.
- static void upwards (Piece *originPiece, Game *game)

  Felfelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void downwards (Piece *originPiece, Game *game)

  Lefelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void rightwards (Piece *originPiece, Game *game)

  Jobbrafelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void leftwards (Piece *originPiece, Game *game)

  Balrafelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void diagonalUpRight (Piece *originPiece, Game *game)

  Északkelet felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void diagonalUpLeft (Piece *originPiece, Game *game)

  Északnyugat felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void diagonalDownRight (Piece *originPiece, Game *game)

  Délkelet felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void diagonalDownLeft (Piece *originPiece, Game *game)

  Délnyugat felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void orthogonal (Piece *originPiece, Game *game)

  Ortogonális lépések(+) Így lép például a bástya. Ez pedig az upwards(),downwards(),rightwards(),leftwards() függvények összesítése.
- static void diagonal (Piece *originPiece, Game *game)

  Átlós lépések(X) Így lép például a futó. Ez pedig a diagonalDownLeft(),diagonalDownRight(),diagonalUpLeft(),diagonalUpRight() függvények összesítése.
- static void pawnMove (Piece *originPiece, Game *game)

Parasztlépés. Algoritmus leírása:

- static void horseMove (Piece *originPiece, Game *game)

  Lólépés. Ló bábú lépéseinek ellenőrzése és hozzáadása a lehetséges lépésekhez.

- static void kingMove (Piece *originPiece, Game *game)

  Királylépés. ellenőrzi a megadott király bábú lehetséges lépéseit a táblán, majd hozzáadja azokat a lehetséges lépések listájához.

Protected Attributes inherited from Piece

- List< Move > piece_moves

4.12.1 Detailed Description

King osztály, ami a királyt reprezentálja a sakkjátékban. Ebből a bábuból minden csapatnak kell, hogy legyen különben automatikusan veszt. Ez a játék feltétele. Ebből lehet több egy csapatban.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 King()

```
King::King (
            int x,
            int y )
```

King osztály konstruktora.

Parameters

x	Az oszlop koordinátája
y	A sor koordinátája

4.12.3 Member Function Documentation

4.12.3.1 calculateMoves()

```
void King::calculateMoves (
            Game * game )  [override], [virtual]
```

Lehetséges lépéseket számolja ki a királynak. A külön erre elkészített kingMove() fv.-t használja.

Parameters

game	A játék objektumra mutató pointer

Implements Piece.

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

- King.h
- King.cpp

4.13 List< T > Class Template Reference

Public Member Functions

- List ()
- ~List ()
- void addtoList (T *newData)
- void deletefromList (T *todelete)
- List (const List &list)
- List & operator= (const List &list)
- int getSize () const
- T * operator[ ] (size_t index) const
- void clear ()
- void consumeList (List &consumed)
- T * Maximum ()

4.13.1 Constructor & Destructor Documentation

4.13.1.1 List() [1/2]

```
template<class T >
List< T >::List ( )  [inline]
```

Konstruktor: inicializálja az alapadatokat.

4.13.1.2 ~List()

```
template<class T >
List< T >::~List ( )  [inline]
```

Destruktor: felszabadítja a lista elemeit.

4.13.1.3 List() [2/2]

```
template<class T >
List< T >::List (
            const List< T > & list )  [inline]
```

Másoló konstruktor: létrehoz egy új listát a paraméterként kapott listából.

4.13.2 Member Function Documentation

4.13.2.1 addtoList()

```
template<class T >
void List< T >::addtoList (
            T * newData )  [inline]
```

Új elem hozzáadása a listához.

4.13.2.2 clear()

```
template<class T >
void List< T >::clear ( )  [inline]
```

Lista elemeinek felszabadítása.

4.13.2.3 consumeList()

```
template<class T >
void List< T >::consumeList (
            List< T > & consumed )  [inline]
```

A lista elemeinek átvétele egy másik listából.(Elfogyasztja a másikat)

4.13.2.4 deletefromList()

```
template<class T >
void List< T >::deletefromList (
            T * todelete )  [inline]
```

Elem törlése a listából.

4.13.2.5 getSize()

```
template<class T >
int List< T >::getSize ( ) const  [inline]
```

Lista méretének lekérése.

4.13.2.6 Maximum()

```
template<class T >
T * List< T >::Maximum ( )  [inline]
```

Lista maximumának keresése.

4.13.2.7 operator=()

```
template<class T >
List & List< T >::operator= (
            const List< T > & list )  [inline]
```

Értékadó operátor: másolja a paraméterként kapott lista elemeit az aktuális listába.

4.13.2.8 operator[]()

```
template<class T >
T * List< T >::operator[] (
            size_t index ) const  [inline]
```

Indexelő operátor: visszaadja az adott indexű elemet.

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

- List.hpp

4.14 Menu Class Reference

Menü class, gombokat tárol.

`#include <menu.h>`

Public Member Functions

- **Menu** ()
  Default konstruktor.
- ~**Menu** ()
  Menu destruktor Törli a dinamikusan foglalt gombtömböt.
- void **incrementCounter** ()
  Inkrementálja az idCounter-t.
- void **updateExit** ()
  Igazzá teszi az exitRequested bool-t és akkor kilép a menüből majd a program.
- void addButton (const Button &extra)
  Hozzáad egy új gombot a menühöz Algoritmus leírása:
- Button * getButton (int idx) const
  Getter függvény a Gombra idx alapján.
- bool getExit () const
  Getter függvény a kilépés bool állapotára.
- size_t getIdCounter () const
  Getter az id adattagra. Az id ilyenkor ebben az esetben már a Menü méretét is számon tartja.

4.14.1 Detailed Description

Menü class, gombokat tárol.

4.14.2 Member Function Documentation

4.14.2.1 addButton()

```
void Menu::addButton (
            const Button & extra )
```

Hozzáad egy új gombot a menühöz Algoritmus leírása:

- Inkrementálja a számlálót
- Lefoglal 1-el több helyet és hozzáadja az új gombot
- Hozzáad

Parameters

extra	Új gomb

4.14.2.2 getButton()

```
Button * Menu::getButton (
            int idx ) const
```

Getter függvény a Gombra idx alapján.

Parameters

idx	sorszám

Returns

buttonTömb[idx]-edik eleme

4.14.2.3 getExit()

```
bool Menu::getExit ( ) const
```

Getter függvény a kilépés bool állapotára.

Returns

true: Ha volt utasítás adva kilépésre

false: Ha nem volt rá utasítás

4.14.2.4 getIdCounter()

```
size_t Menu::getIdCounter ( ) const
```

Getter az id adattagra. Az id ilyenkor ebben az esetben már a Menü méretét is számon tartja.

Returns

Menü mérete

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

- menu.h
- menu.cpp

4.15 Move Class Reference

A Move class. Ezt tárolják el a bábuk, emellett csapatok listákban.

```
#include <Move.h>
```

Public Member Functions

- Move (Piece *originPiece, int coordX, int coordY, char destinationPieceName=0)

  Move konstruktor Miután inicializál minden értéket(kivéve Weight) lefut benne a calculateWeight() amiből megkapja a súlyozását.
- ~**Move** ()

  Move destruktor.
- void **calculateWeight** ()

  Kiszámolja egy Move objektum súlyát. Úgy számolja ki, hogy minél értékesebb a bábu a célmezőn annál nagyobb súlyt ad.
- bool operator> (const Move &otherMove) const

  Nagyobb operátor Eldönti, hogy nagyobb-e jobboldali Move-nál nagyobb súly alapján.
- Piece * getPiece () const

  Getter függvény az originPiece-re. Rámutat arra a Piece-re akiből származik.
- int getCoordX () const

  Getter függvény x koordinátára.
- int getCoordY () const

  Getter függvény y koordinátára.
- int getWeight () const

  Getter függvény a lépés súlyára.

4.15.1 Detailed Description

A Move class. Ezt tárolják el a bábuk, emellett csapatok listákban.

4.15.2 Constructor & Destructor Documentation

4.15.2.1 Move()

```
Move::Move (
            Piece * originPiece,
            int coordX,
            int coordY,
            char destinationPieceName = 0 )
```

Move konstruktor Miután inicializál minden értéket(kivéve Weight) lefut benne a calculateWeight() amiből megkapja a súlyozását.

Parameters

originPiece	A bábu aki meg tudja tenni a lépést.
coordX	Az adott mező x koordinátája.
coordY	Az adott mező y koordinátája.
destinationPieceName	Az adott mezőn elhelyezkedő ellenséges bábu.

4.15.3 Member Function Documentation

4.15.3.1 getCoordX()

```
int Move::getCoordX ( ) const
```

Getter függvény x koordinátára.

Returns

x koordináta

4.15.3.2 getCoordY()

```
int Move::getCoordY ( ) const
```

Getter függvény y koordinátára.

Returns

y koordináta

4.15.3.3 getPiece()

```
Piece * Move::getPiece ( ) const
```

Getter függvény az originPiece-re. Rámutat arra a Piece-re akiből származik.

Returns

Piece∗

4.15.3.4 getWeight()

```
int Move::getWeight ( ) const
```

Getter függvény a lépés súlyára.

Returns

Lépés súlya

4.15.3.5 operator>()

```
bool Move::operator> (
            const Move & otherMove ) const
```

Nagyobb operátor Eldönti, hogy nagyobb-e jobboldali Move-nál nagyobb súly alapján.

Parameters

otherMove	Jobb oldali Move érték

Returns

true: Bal oldali érték nagyobb

false: Jobb oldali érték nagyobb

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

- Move.h
- Move.cpp

4.16 Node< T > Struct Template Reference

Public Member Functions

- Node (T *newData)
- ~Node ()
- T * getData ()
- T * release ()

Public Attributes

- T * data
- Node< T > * previous
- Node< T > * next

4.16.1 Constructor & Destructor Documentation

4.16.1.1 Node()

```
template<class T >
Node< T >::Node (
            T * newData )  [inline]
```

Konstruktor

4.16.1.2 ~Node()

```
template<class T >
Node< T >::~Node ( )  [inline]
```

Destruktor, felszabadítja az adatot

4.16.2 Member Function Documentation

4.16.2.1 getData()

```
template<class T >
T * Node< T >::getData ( )  [inline]
```

< Adat lekérése

4.16.2.2 release()

```
template<class T >
T * Node< T >::release ( )  [inline]
```

< Adat felszabadítása és visszaadása

4.16.3 Member Data Documentation

4.16.3.1 data

```
template<class T >
T* Node< T >::data
```

Adat pointere

4.16.3.2 next

```
template<class T >
Node<T>* Node< T >::next
```

Következő Node-ra mutató pointer

4.16.3.3 previous

```
template<class T >
Node<T>* Node< T >::previous
```

Előző Node-ra mutató pointer

The documentation for this struct was generated from the following file:

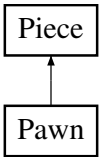
- List.hpp

4.17 Pawn Class Reference

Pawn osztály, ami a gyalogot reprezentálja a sakkjátékban.

`#include <Pawn.h>`

Inheritance diagram for Pawn:



Public Member Functions

- Pawn (int x, int y)

  Pawn osztály konstruktora.
- void calculateMoves (Game *game) override

  Lehetséges lépéseket számolja ki a gyalognak. A külön erre elkészített pawnMove() fv.-t használja.

Public Member Functions inherited from Piece

- Piece (char name, int coordX=0, int coordY=0)

  Bábu konstruktor.
- virtual ~**Piece** ()

  Bábu destruktor. Amikor lefut akkor felszabadul a listája ami tárolja a lehetséges lépéseit(Persze, csak ha játékban van, mivel csak akkor vannak számolva a lépései).
- int getCoordY () const

  Getter függvény az y koordinátára.
- int getCoordX () const

  Getter függvény az x koordinátára.
- char getName () const

  Getter függvény a bábu nevére.
- List< Move > & getMoves ()

  Getter függvény a bábu lépéseinek a listájára.
- void setCoordY (int newY)

  Setter függvény a bábu Y koordinátájának.
- void setCoordX (int newX)

  Setter függvény a bábu x koordinátájának.
- bool operator== (const Piece &otherPiece) const

  Egyenlőség operátor.
- void **toLowercase** ()

  Átváltja egy bábunak a nevét kisbetűsre nagybetűsről. Ezt a tükrözésnél használja a program.
- void addMove (char destinationPieceName, int coordX, int coordY)

  Hozzáad egy dinamikusan foglalt Move objektumot a meglévő Piecenek a Move listájához. Ezt a függvényt majd a checkAndAddMove(...) használja majd.

Additional Inherited Members

Static Public Member Functions inherited from Piece

- static Piece * createPiece (char name, int x, int y)

  Létrehoz egy Piece objektumot. Név alapján hozza létre a bábut. Ha rossz nevet kap, akkor nullptr-t ad vissza.
- static bool checkAndAddMove (Game *game, Piece *originPiece, int posX, int posY)

  Kiszámolja, hogy adott mezőre léphet-e a Bábu. Ha igen akkor hozzáadja a lépéseihez. Emellett visszatér egy bool value-val, mivel a legtöbb bábu egy hosszabb vonalban lép.
- static void upwards (Piece *originPiece, Game *game)

  Felfelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void downwards (Piece *originPiece, Game *game)

  Lefelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void rightwards (Piece *originPiece, Game *game)

  Jobbrafelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void leftwards (Piece *originPiece, Game *game)

  Balrafelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void diagonalUpRight (Piece *originPiece, Game *game)

  Északkelet felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void diagonalUpLeft (Piece *originPiece, Game *game)

  Északnyugat felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void diagonalDownRight (Piece *originPiece, Game *game)

  Délkelet felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void diagonalDownLeft (Piece *originPiece, Game *game)

  Délnyugat felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void orthogonal (Piece *originPiece, Game *game)

  Ortogonális lépések(+) Így lép például a bástya. Ez pedig az upwards(),downwards(),rightwards(),leftwards() függvények összesítése.
- static void diagonal (Piece *originPiece, Game *game)

  Átlós lépések(X) Így lép például a futó. Ez pedig a diagonalDownLeft(),diagonalDownRight(),diagonalUpLeft(),diagonalUpRight() függvények összesítése.
- static void pawnMove (Piece *originPiece, Game *game)

  Parasztlépés. Algoritmus leírása:
- static void horseMove (Piece *originPiece, Game *game)

  Lólépés. Ló bábú lépéseinek ellenőrzése és hozzáadása a lehetséges lépésekhez.
- static void kingMove (Piece *originPiece, Game *game)

  Királylépés. ellenőrzi a megadott király bábú lehetséges lépéseit a táblán, majd hozzáadja azokat a lehetséges lépések listájához.

Protected Attributes inherited from Piece

- List< Move > piece_moves

4.17.1 Detailed Description

Pawn osztály, ami a gyalogot reprezentálja a sakkjátékban.

4.17.2 Constructor & Destructor Documentation

4.17.2.1 Pawn()

```
Pawn::Pawn (
            int x,
            int y )
```

Pawn osztály konstruktora.

Parameters

x	Az oszlop koordinátája
y	A sor koordinátája

4.17.3 Member Function Documentation

4.17.3.1 calculateMoves()

```
void Pawn::calculateMoves (
            Game * game )  [override], [virtual]
```

Lehetséges lépéseket számolja ki a gyalognak. A külön erre elkészített pawnMove() fv.-t használja.

Parameters

game	A játék objektumra mutató pointer

Implements Piece.

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

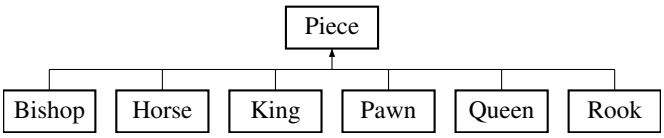
- Pawn.h
- Pawn.cpp

4.18 Piece Class Reference

Bábu osztály. Ez egy absztrakt osztály, amiből az összes sakkbábu osztály öröklődik. Belőlük épülnek fel a seregek.

```
#include <Piece.h>
```

Inheritance diagram for Piece:



Public Member Functions

- Piece (char name, int coordX=0, int coordY=0)

  Bábu konstruktor.
- virtual ~**Piece** ()

  Bábu destruktor. Amikor lefut akkor felszabadul a listája ami tárolja a lehetséges lépéseit(Persze, csak ha játékban van, mivel csak akkor vannak számolva a lépései).
- int getCoordY () const

  Getter függvény az y koordinátára.
- int getCoordX () const

  Getter függvény az x koordinátára.
- char getName () const

  Getter függvény a bábu nevére.
- List< Move > & getMoves ()

  Getter függvény a bábu lépéseinek a listájára.
- void setCoordY (int newY)

  Setter függvény a bábu Y koordinátájának.
- void setCoordX (int newX)

  Setter függvény a bábu x koordinátájának.
- bool operator== (const Piece &otherPiece) const

  Egyenlőség operátor.
- void **toLowercase** ()

  Átváltja egy bábunak a nevét kisbetűsre nagybetűsről. Ezt a tükrözésnél használja a program.
- virtual void calculateMoves (Game *game)=0

  Kiszámolja egy bábu lépéseit Viszont tisztán virtuális, tehát majd az ebből a classból öröklődő osztályok fogják megvalósítani.
- void addMove (char destinationPieceName, int coordX, int coordY)

  Hozzáad egy dinamikusan foglalt Move objektumot a meglévő Piecenek a Move listájához. Ezt a függvényt majd a checkAndAddMove(...) használja majd.

Static Public Member Functions

- static Piece * createPiece (char name, int x, int y)

  Létrehoz egy Piece objektumot. Név alapján hozza létre a bábut. Ha rossz nevet kap, akkor nullptr-t ad vissza.
- static bool checkAndAddMove (Game *game, Piece *originPiece, int posX, int posY)

  Kiszámolja, hogy adott mezőre léphet-e a Bábu. Ha igen akkor hozzáadja a lépéseihez. Emellett visszatér egy bool value-val, mivel a legtöbb bábu egy hosszabb vonalban lép.
- static void upwards (Piece *originPiece, Game *game)

  Felfelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void downwards (Piece *originPiece, Game *game)

  Lefelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void rightwards (Piece *originPiece, Game *game)

Jobbrafelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.

- static void leftwards (Piece ∗originPiece, Game ∗game)

  Balrafelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void diagonalUpRight (Piece ∗originPiece, Game ∗game)

  Északkelet felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void diagonalUpLeft (Piece ∗originPiece, Game ∗game)

  Északnyugat felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void diagonalDownRight (Piece ∗originPiece, Game ∗game)

  Délkelet felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void diagonalDownLeft (Piece ∗originPiece, Game ∗game)

  Délnyugat felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void orthogonal (Piece ∗originPiece, Game ∗game)

  Ortogonális lépések(+) Így lép például a bástya. Ez pedig az upwards(),downwards(),rightwards(),leftwards() függvények összesítése.
- static void diagonal (Piece ∗originPiece, Game ∗game)

  Átlós lépések(X) Így lép például a futó. Ez pedig a diagonalDownLeft(),diagonalDownRight(),diagonalUpLeft(),diagonalUpRight() függvények összesítése.
- static void pawnMove (Piece ∗originPiece, Game ∗game)

  Parasztlépés. Algoritmus leírása:
- static void horseMove (Piece ∗originPiece, Game ∗game)

  Lólépés. Ló bábú lépéseinek ellenőrzése és hozzáadása a lehetséges lépésekhez.
- static void kingMove (Piece ∗originPiece, Game ∗game)

  Királylépés. ellenőrzi a megadott király bábú lehetséges lépéseit a táblán, majd hozzáadja azokat a lehetséges lépések listájához.

Protected Attributes

- List< Move > piece_moves

4.18.1 Detailed Description

Bábu osztály. Ez egy absztrakt osztály, amiből az összes sakkbábu osztály öröklődik. Belőlük épülnek fel a seregek.

4.18.2 Constructor & Destructor Documentation

4.18.2.1 Piece()

```
Piece::Piece (
            char name,
            int coordX = 0,
            int coordY = 0 )
```

Bábu konstruktor.

Parameters

name	Bábu neve.
coordX	Bábu x koordinátája.
coordY	Bábu y koordinátája.

4.18.3 Member Function Documentation

4.18.3.1 addMove()

```
void Piece::addMove (
            char destinationPieceName,
            int coordX,
            int coordY )
```

Hozzáad egy dinamikusan foglalt Move objektumot a meglévő Piecenek a Move listájához. Ezt a függvényt majd a checkAndAddMove(...) használja majd.

Parameters

destinationPieceName	A célmezőn lévő bábu neve.
coordX	A célmező x koordinátája.
coordY	A célmező Y koordinátája.

4.18.3.2 calculateMoves()

```
virtual void Piece::calculateMoves (
            Game * game )  [pure virtual]
```

Kiszámolja egy bábu lépéseit Viszont tisztán virtuális, tehát majd az ebből a classból öröklődő osztályok fogják megvalósítani.

Parameters

game	Játék objektumra mutató pointer.

Implemented in Bishop, Horse, King, Pawn, Queen, and Rook.

4.18.3.3 checkAndAddMove()

```
bool Piece::checkAndAddMove (
            Game * game,
            Piece * originPiece,
            int posX,
            int posY )  [static]
```

Kiszámolja, hogy adott mezőre léphet-e a Bábu. Ha igen akkor hozzáadja a lépéseihez. Emellett visszatér egy bool value-val, mivel a legtöbb bábu egy hosszabb vonalban lép.

- Ha hamissal tér vissza, akkor megszakad a vonal amiben tud lépni. Ami azt jelenti, hogy a vonal útját állja valami. És a ló kivételével nem is léphet át semmi a másikon.

Parameters

game	Game objektum pointer.
originPiece	A bábu akiből a lépés származik.
posX	A célmező x koordinátája.
posY	A célmező y koordinátája.

Returns

true: Elállta valami az útját.(lehet az ellenség[akkor leüthető] vagy csapattárs[akkor nem üthető]) Ha ciklusban volt nem mehet tovább.

false: Nem állta semmi az útját. Ha ciklusban van mehet tovább.

4.18.3.4 createPiece()

```
Piece * Piece::createPiece (
            char name,
            int x,
            int y )  [static]
```

Létrehoz egy Piece objektumot. Név alapján hozza létre a bábut. Ha rossz nevet kap, akkor nullptr-t ad vissza.

Parameters

name	Létrehozandó bábu neve. Ez alapján dönti el, hogy milyen konstruktort futtasson le a bábunak.
x	A bábu x koordinátája.
y	A bábu y koordinátája.

Returns

Létrehozott bábura mutató pointer.

4.18.3.5 diagonal()

```
void Piece::diagonal (
            Piece * originPiece,
            Game * game )  [static]
```

Átlós lépések(X) Így lép például a futó. Ez pedig a diagonalDownLeft(),diagonalDownRight(),diagonalUpLeft(),diagonalUpRight() függvények összesítése.

Parameters

originPiece	A bábu akiből származik a lépés.
game	Game objektumra mutató pointer.

4.18.3.6 diagonalDownLeft()

```
void Piece::diagonalDownLeft (
            Piece * originPiece,
            Game * game )  [static]
```

Délnyugat felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.

Parameters

originPiece	A bábu akiből származik a lépés.
game	Game objektumra mutató pointer.

4.18.3.7 diagonalDownRight()

```
void Piece::diagonalDownRight (
            Piece * originPiece,
            Game * game )  [static]
```

Délkelet felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.

Parameters

originPiece	A bábu akiből származik a lépés.
game	Game objektumra mutató pointer.

4.18.3.8 diagonalUpLeft()

```
void Piece::diagonalUpLeft (
            Piece * originPiece,
            Game * game )  [static]
```

Északnyugat felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.

Parameters

originPiece	A bábu akiből származik a lépés.
game	Game objektumra mutató pointer.

4.18.3.9 diagonalUpRight()

```
void Piece::diagonalUpRight (
            Piece * originPiece,
            Game * game )  [static]
```

Északkelet felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.

Parameters

originPiece	A bábu akiből származik a lépés.
game	Game objektumra mutató pointer.

4.18.3.10 downwards()

```
void Piece::downwards (
            Piece * originPiece,
            Game * game )  [static]
```

Lefelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.

Parameters

originPiece	A bábu akiből származik a lépés.
game	Game objektumra mutató pointer.

4.18.3.11 getCoordX()

```
int Piece::getCoordX ( ) const
```

Getter függvény az x koordinátára.

Returns

Bábu x koordinátája.

4.18.3.12 getCoordY()

```
int Piece::getCoordY ( ) const
```

Getter függvény az y koordinátára.

Returns

Bábu y koordinátája.

4.18.3.13 getMoves()

```
List< Move > & Piece::getMoves ( )
```

Getter függvény a bábu lépéseinek a listájára.

Returns

Bábu lépéseinek listája.

4.18.3.14 getName()

```
char Piece::getName ( ) const
```

Getter függvény a bábu nevére.

Returns

Bábu neve.

4.18.3.15 horseMove()

```
void Piece::horseMove (
            Piece * originPiece,
            Game * game )  [static]
```

Lólépés. Ló bábú lépéseinek ellenőrzése és hozzáadása a lehetséges lépésekhez.

Parameters

originPiece	A bábu akiből származik a lépés.
game	Game objektumra mutató pointer.

4.18.3.16 kingMove()

```
void Piece::kingMove (
            Piece * originPiece,
            Game * game )  [static]
```

Királylépés. ellenőrzi a megadott király bábú lehetséges lépéseit a táblán, majd hozzáadja azokat a lehetséges lépések listájához.

Parameters

originPiece	A bábu akiből származik a lépés.
game	Game objektumra mutató pointer.

4.18.3.17 leftwards()

```
void Piece::leftwards (
            Piece * originPiece,
            Game * game )  [static]
```

Balrafelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.

Parameters

originPiece	A bábu akiből származik a lépés.
game	Game objektumra mutató pointer.

4.18.3.18 operator==()

```
bool Piece::operator== (
            const Piece & otherPiece ) const
```

Egyenlőség operátor.

Parameters

otherPiece	Jobboldali Piece referencia

Returns

true: Ha ugyanazon a mezőn vannak akkor igazat ad.

false: Ha különböző mezőkön állnak akkor hamis.

4.18.3.19 orthogonal()

```
void Piece::orthogonal (
            Piece * originPiece,
            Game * game )  [static]
```

Ortogonális lépések(+) Így lép például a bástya. Ez pedig az upwards(),downwards(),rightwards(),leftwards() függvények összesítése.

Parameters

originPiece	A bábu akiből származik a lépés.
game	Game objektumra mutató pointer.

4.18.3.20 pawnMove()

```
void Piece::pawnMove (
            Piece * originPiece,
            Game * game )  [static]
```

Parasztlépés. Algoritmus leírása:

- Először eldöntjük, hogy melyik színben vagyunk, mivel a fekete parasztok csak lefele, a fehér pedig felfele tud lépni
- A paraszt előtti mezőt hozzáadjuk a lépéseihez, ha nincsen elfoglalva

- Majd megnézzük, hogy az amellett lévő 2 szomszédos mező üthető-e egyébként, ha igen, akkor hozzáadjuk a lépéseihez.

Parameters

originPiece	A bábu akiből származik a lépés.
game	Game objektumra mutató pointer.

4.18.3.21 rightwards()

```
void Piece::rightwards (
            Piece * originPiece,
            Game * game )  [static]
```

Jobbrafelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.

Parameters

originPiece	A bábu akiből származik a lépés.
game	Game objektumra mutató pointer.

4.18.3.22 setCoordX()

```
void Piece::setCoordX (
            int newX )
```

Setter függvény a bábu x koordinátájának.

Parameters

newX	Új x koordináta érték.

4.18.3.23 setCoordY()

```
void Piece::setCoordY (
            int newY )
```

Setter függvény a bábu Y koordinátájának.

Parameters

newY	Új y koordináta érték.

4.18.3.24 upwards()

```
void Piece::upwards (
```

```
Piece * originPiece,
Game * game )  [static]
```

Felfelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.

Parameters

originPiece	A bábu akiből származik a lépés.
game	Game objektumra mutató pointer.

4.18.4 Member Data Documentation

4.18.4.1 piece_moves

```
List<Move> Piece::piece_moves  [protected]
```

A bábu lépéseit tároló lista

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

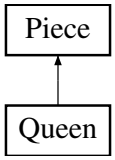
- Piece.h
- Piece.cpp

4.19 Queen Class Reference

Queen osztály, ami a vezért reprezentálja a sakkjátékban.

```
#include <Queen.h>
```

Inheritance diagram for Queen:



Public Member Functions

- Queen (int x, int y)

  Queen osztály konstruktora.
- void calculateMoves (Game *game) override

  Lehetséges lépéseket számolja ki a vezérnek. Ezek a lépések együtt az orthogonális és az átlós lépések kombinálva.

Public Member Functions inherited from Piece

- Piece (char name, int coordX=0, int coordY=0)

  Bábu konstruktor.
- virtual ~**Piece** ()

  Bábu destruktor. Amikor lefut akkor felszabadul a listája ami tárolja a lehetséges lépéseit(Persze, csak ha játékban van, mivel csak akkor vannak számolva a lépései).
- int getCoordY () const

  Getter függvény az y koordinátára.
- int getCoordX () const

  Getter függvény az x koordinátára.
- char getName () const

  Getter függvény a bábu nevére.
- List< Move > & getMoves ()

  Getter függvény a bábu lépéseinek a listájára.
- void setCoordY (int newY)

  Setter függvény a bábu Y koordinátájának.
- void setCoordX (int newX)

  Setter függvény a bábu x koordinátájának.
- bool operator== (const Piece &otherPiece) const

  Egyenlőség operátor.
- void **toLowercase** ()

  Átváltja egy bábunak a nevét kisbetűsre nagybetűsről. Ezt a tükrözésnél használja a program.
- void addMove (char destinationPieceName, int coordX, int coordY)

  Hozzáad egy dinamikusan foglalt Move objektumot a meglévő Piecenek a Move listájához. Ezt a függvényt majd a checkAndAddMove(...) használja majd.

Additional Inherited Members

Static Public Member Functions inherited from Piece

- static Piece * createPiece (char name, int x, int y)

  Létrehoz egy Piece objektumot. Név alapján hozza létre a bábut. Ha rossz nevet kap, akkor nullptr-t ad vissza.
- static bool checkAndAddMove (Game *game, Piece *originPiece, int posX, int posY)

  Kiszámolja, hogy adott mezőre léphet-e a Bábu. Ha igen akkor hozzáadja a lépéseihez. Emellett visszatér egy bool value-val, mivel a legtöbb bábu egy hosszabb vonalban lép.
- static void upwards (Piece *originPiece, Game *game)

  Felfelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void downwards (Piece *originPiece, Game *game)

  Lefelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void rightwards (Piece *originPiece, Game *game)

  Jobbrafelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void leftwards (Piece *originPiece, Game *game)

  Balrafelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void diagonalUpRight (Piece *originPiece, Game *game)

  Északkelet felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void diagonalUpLeft (Piece *originPiece, Game *game)

Északnyugat felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.

- static void diagonalDownRight (Piece *originPiece, Game *game)

  Délkelet felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void diagonalDownLeft (Piece *originPiece, Game *game)

  Délnyugat felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void orthogonal (Piece *originPiece, Game *game)

  Ortogonális lépések(+) Így lép például a bástya. Ez pedig az upwards(),downwards(),rightwards(),leftwards() függvények összesítése.
- static void diagonal (Piece *originPiece, Game *game)

  Átlós lépések(X) Így lép például a futó. Ez pedig a diagonalDownLeft(),diagonalDownRight(),diagonalUpLeft(),diagonalUpRight() függvények összesítése.
- static void pawnMove (Piece *originPiece, Game *game)

  Parasztlépés. Algoritmus leírása:
- static void horseMove (Piece *originPiece, Game *game)

  Lólépés. Ló bábú lépéseinek ellenőrzése és hozzáadása a lehetséges lépésekhez.
- static void kingMove (Piece *originPiece, Game *game)

  Királylépés. ellenőrzi a megadott király bábú lehetséges lépéseit a táblán, majd hozzáadja azokat a lehetséges lépések listájához.

Protected Attributes inherited from Piece

- List< Move > piece_moves

4.19.1 Detailed Description

Queen osztály, ami a vezért reprezentálja a sakkjátékban.

4.19.2 Constructor & Destructor Documentation

4.19.2.1 Queen()

```
Queen::Queen (
            int x,
            int y )
```

Queen osztály konstruktora.

Parameters

x	Az oszlop koordinátája
y	A sor koordinátája

4.19.3 Member Function Documentation

4.19.3.1 calculateMoves()

```
void Queen::calculateMoves (
            Game * game )  [override], [virtual]
```

Lehetséges lépéseket számolja ki a vezérnek. Ezek a lépések együtt az orthogonális és az átlós lépések kombinálva.

Parameters

game	A játék objektumra mutató pointer

Implements Piece.

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

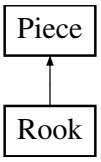
- Queen.h
- Queen.cpp

4.20 Rook Class Reference

Rook osztály, ami a bástyát reprezentálja a sakkjátékban.

```
#include <Rook.h>
```

Inheritance diagram for Rook:



Public Member Functions

- Rook (int x, int y)

  Rook osztály konstruktora.
- void calculateMoves (Game *game) override

  Lehetséges lépéseket számolja ki a bástyának. Tehát belerakja az orthogonális lépéseket kiszámoló függvényt.

Public Member Functions inherited from Piece

- Piece (char name, int coordX=0, int coordY=0)

  Bábu konstruktor.
- virtual ~**Piece** ()

  Bábu destruktor. Amikor lefut akkor felszabadul a listája ami tárolja a lehetséges lépéseit(Persze, csak ha játékban van, mivel csak akkor vannak számolva a lépései).
- int getCoordY () const

  Getter függvény az y koordinátára.
- int getCoordX () const

  Getter függvény az x koordinátára.
- char getName () const

  Getter függvény a bábu nevére.
- List< Move > & getMoves ()

  Getter függvény a bábu lépéseinek a listájára.
- void setCoordY (int newY)

  Setter függvény a bábu Y koordinátájának.
- void setCoordX (int newX)

  Setter függvény a bábu x koordinátájának.
- bool operator== (const Piece &otherPiece) const

  Egyenlőség operátor.
- void **toLowercase** ()

  Átváltja egy bábunak a nevét kisbetűsre nagybetűsről. Ezt a tükrözésnél használja a program.
- void addMove (char destinationPieceName, int coordX, int coordY)

  Hozzáad egy dinamikusan foglalt Move objektumot a meglévő Piecenek a Move listájához. Ezt a függvényt majd a checkAndAddMove(...) használja majd.

Additional Inherited Members

Static Public Member Functions inherited from Piece

- static Piece * createPiece (char name, int x, int y)

  Létrehoz egy Piece objektumot. Név alapján hozza létre a bábut. Ha rossz nevet kap, akkor nullptr-t ad vissza.
- static bool checkAndAddMove (Game *game, Piece *originPiece, int posX, int posY)

  Kiszámolja, hogy adott mezőre léphet-e a Bábu. Ha igen akkor hozzáadja a lépéseihez. Emellett visszatér egy bool value-val, mivel a legtöbb bábu egy hosszabb vonalban lép.
- static void upwards (Piece *originPiece, Game *game)

  Felfelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void downwards (Piece *originPiece, Game *game)

  Lefelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void rightwards (Piece *originPiece, Game *game)

  Jobbrafelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void leftwards (Piece *originPiece, Game *game)

  Balrafelé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void diagonalUpRight (Piece *originPiece, Game *game)

  Északkelet felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void diagonalUpLeft (Piece *originPiece, Game *game)

Északnyugat felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.

- static void diagonalDownRight (Piece *originPiece, Game *game)

  Délkelet felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void diagonalDownLeft (Piece *originPiece, Game *game)

  Délnyugat felé egy vonalban az összes lépés a bábu pozíciójától. Addig megy amíg el nem állja valami az útját vagy ki nem indexel.
- static void orthogonal (Piece *originPiece, Game *game)

  Ortogonális lépések(+) Így lép például a bástya. Ez pedig az upwards(),downwards(),rightwards(),leftwards() függvények összesítése.
- static void diagonal (Piece *originPiece, Game *game)

  Átlós lépések(X) Így lép például a futó. Ez pedig a diagonalDownLeft(),diagonalDownRight(),diagonalUpLeft(),diagonalUpRight() függvények összesítése.
- static void pawnMove (Piece *originPiece, Game *game)

  Parasztlépés. Algoritmus leírása:
- static void horseMove (Piece *originPiece, Game *game)

  Lólépés. Ló bábú lépéseinek ellenőrzése és hozzáadása a lehetséges lépésekhez.
- static void kingMove (Piece *originPiece, Game *game)

  Királylépés. ellenőrzi a megadott király bábú lehetséges lépéseit a táblán, majd hozzáadja azokat a lehetséges lépések listájához.

Protected Attributes inherited from Piece

- List< Move > piece_moves

4.20.1 Detailed Description

Rook osztály, ami a bástyát reprezentálja a sakkjátékban.

4.20.2 Constructor & Destructor Documentation

4.20.2.1 Rook()

```
Rook::Rook (
            int x,
            int y )
```

Rook osztály konstruktora.

Parameters

x	Az oszlop koordinátája
y	A sor koordinátája

4.20.3 Member Function Documentation

4.20.3.1 calculateMoves()

```
void Rook::calculateMoves (
            Game * game )  [override], [virtual]
```

Lehetséges lépéseket számolja ki a bástyának. Tehát belerakja az orthogonális lépéseket kiszámoló függvényt.

Parameters

game	A játék objektumra mutató pointer

Implements Piece.

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

- Rook.h
- Rook.cpp

4.21 Team Class Reference

A Team class. Ebből jön létre 2 db amikor elindul a játék-szimuláció. Ide vannak kigyűjtve a seregek lépései miután a bábuk összegyűjtötték a lehetséges lépéseiket.

`#include <Team.h>`

Public Member Functions

- Team (Army *army, TeamColor color)

  Team konstruktor Ha a színe fekete, akkor a bábuit feltükrözi. A seregre a helyet dinamikusan lefoglalja és átmásolja oda a paraméterként kapott Armyból a bábukat.
- ~**Team** ()

  Team destruktor Felszabadítja a dinamikusan lefoglalt seregét.
- Army * getArmy ()

  Getter függvény a csapaton belüli seregre.
- List< Move > & getTeamMoves ()

  Getter függvény a csapat összesített lépéseinek listájára.
- Move * getRandomMove ()

  Random lépést kiválaszt a lépések listájából, ha nincsen maximum.
- int countAmountOfKigns ()

  Megszámolja azt, hogy hány király van egy csapatban. Azért fontos, mert ha 0 van egy csapatban akkor a csapat vesztett. Ez minden körben lefut.

4.21.1 Detailed Description

A Team class. Ebből jön létre 2 db amikor elindul a játék-szimuláció. Ide vannak kigyűjtve a seregek lépései miután a bábuk összegyűjtötték a lehetséges lépéseiket.

4.21.2 Constructor & Destructor Documentation

4.21.2.1 Team()

```
Team::Team (
            Army * army,
            TeamColor color )
```

Team konstruktor Ha a színe fekete, akkor a bábuit feltükrözi. A seregre a helyet dinamikusan lefoglalja és átmásolja oda a paraméterként kapott Armyból a bábukat.

Parameters

army	A sereg amivel a csapat fog játszani.
color	A csapat színe.

4.21.3 Member Function Documentation

4.21.3.1 countAmountOfKigns()

```
int Team::countAmountOfKigns ( )
```

Megszámolja azt, hogy hány király van egy csapatban. Azért fontos, mert ha 0 van egy csapatban akkor a csapat vesztett. Ez minden körben lefut.

Returns

Királyok darabaszáma egy csapatban.

4.21.3.2 getArmy()

```
Army * Team::getArmy ( )
```

Getter függvény a csapaton belüli seregre.

Returns

Seregére mutató pointer

4.21.3.3 getRandomMove()

```
Move * Team::getRandomMove ( )
```

Random lépést kiválaszt a lépések listájából, ha nincsen maximum.

Returns

Random lépés.

4.21.3.4 getTeamMoves()

```
List< Move > & Team::getTeamMoves ( )
```

Getter függvény a csapat összesített lépéseinek listájára.

Returns

Összesített lépések listája.

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

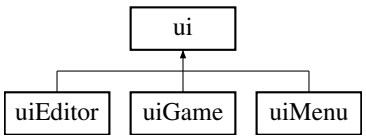
- Team.h
- Team.cpp

4.22 ui Class Reference

ui absztakt osztály Ebből származik a uiGame, uiEditor, uiMenu. Ezek az osztályok tartják fent a kapcsolatot a felhasználó és számítógép között.

```
#include <ui.h>
```

Inheritance diagram for ui:



Public Member Functions

- ui ()
- virtual void display ()=0

  Tisztán virtuális tag függvény a képernyőre való megjelenítésért. Ez mutatja meg a felhasználónak a program válaszreakcióját a felhasználó bemenetére.
- virtual bool handleInput ()=0
- virtual void idle ()=0

  Tisztán virtuális tagfgv. Ez felelős a display() és handleInput() egymással való kommunkiációjának kialakítására. Minden ebből származó osztályban az idle() úgy néz ki, hogy egymás után következik a display() és handleInput() és ezek vannak egy while ciklusban amiből akkor lép ki, ha parancsot kapott a kilépésre a handleInput()-on keresztül.

Static Public Member Functions

- static void **clearScreen** ()

  Letisztítja a képernyőt. Letörli a képernyőn kiírt adatokat, hogy az újakat "tiszta lapra" lehessen kiírni. A tagfüggvény van arra is kezelve ha windows-ról van futtatva vagy linuxról.
- static void delayMilliseconds (unsigned int ms)

  Függvény adott milliszekundum-nyi késleltetésre. Hasznos arra, hogy egy sakk meccs ne egyből fusson le, hanem mindig várjon a gép lépésenként, hogy legyen ideje felfognia a felhasználónak. A tagfüggvény van arra is kezelve ha windows-ról van futtatva vagy linuxról.

4.22.1 Detailed Description

ui absztakt osztály Ebből származik a uiGame, uiEditor, uiMenu. Ezek az osztályok tartják fent a kapcsolatot a felhasználó és számítógép között.

4.22.2 Constructor & Destructor Documentation

4.22.2.1 ui()

```
ui::ui ( )  [inline]
```

Default konstruktor

4.22.3 Member Function Documentation

4.22.3.1 delayMilliseconds()

```
void ui::delayMilliseconds (
            unsigned int ms )  [static]
```

Függvény adott milliszekundum-nyi késleltetésre. Hasznos arra, hogy egy sakk meccs ne egyből fusson le, hanem mindig várjon a gép lépésenként, hogy legyen ideje felfognia a felhasználónak. A tagfüggvény van arra is kezelve ha windows-ról van futtatva vagy linuxról.

Parameters

ms	milliszekundum-nyi késleltetés

4.22.3.2 display()

```
virtual void ui::display ( )  [pure virtual]
```

Tisztán virtuális tag függvény a képernyőre való megjelenítésért. Ez mutatja meg a felhasználónak a program válaszreakcióját a felhasználó bemenetére.

Implemented in uiEditor, uiGame, and uiMenu.

4.22.3.3 handleInput()

```
virtual bool ui::handleInput ( )  [pure virtual]
```

Tisztán virtuális tagfgv. Ez felelős a bemenetek kezeléséért.

Implemented in uiEditor, uiGame, and uiMenu.

4.22.3.4 idle()

```
virtual void ui::idle ( )  [pure virtual]
```

Tisztán virtuális tagfgv. Ez felelős a display() és handleInput() egymással való kommunkiációjának kialakítására. Minden ebből származó osztályban az idle() úgy néz ki, hogy egymás után következik a display() és handleInput() és ezek vannak egy while ciklusban amiből akkor lép ki, ha parancsot kapott a kilépésre a handleInput()-on keresztül.

Implemented in uiEditor, uiGame, and uiMenu.

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

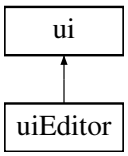
- ui.h
- ui.cpp

4.23 uiEditor Class Reference

uiEditor osztály, ami az editor megjelenítéséért felelős.

```
#include <uiEditor.h>
```

Inheritance diagram for uiEditor:



Public Member Functions

- uiEditor (Editor *editor)
- ~uiEditor ()
- void display () override

  A szerkesztő megjelenítőfüggvénye. Ez 2 másik függvényt foglal magába.
- bool handleInput () override

  A szerkesztőnek az inputkezelésével foglalkozó függvény.
- void **renderTable** ()

  A tábla lerenderéléséért felelős függvény. Ez csak egy fél táblát fog kiírni, hiszen csak 1 térfelet akarunk szerkeszteni nem többet. Emellett minden mezőben leellenőrzi, hogy lenne-e benne bábu, mert ha igen akkor beleírja a bábu betűjelét. Egyébként meg üresen hagyja.
- void idle () override

  A szerkesztő inaktív állapota lényegében.(várja a választ a felektől egymás fele) Felhasználó és program közötti kommunikáció a szerkesztőben amik kilépésre nem kap jelet a program.
- void **saveSequence** ()

  Újonnan lementett seregek mentésére szolgáló protokol. Ha a felhasználó egy újonnan létrehozott sereget el szeretne menteni, akkor kilépéskor ezen keresztül tudja kimenteni a fájlba.

Public Member Functions inherited from ui

- ui ()

Static Public Member Functions

- static void Run (Editor *editorptr)

  Futtatja a paraméterként kapott szerkesztőt Létrehoz neki egy uiEditor-t és azt berakja idle() pozícióba. A fv. tudni fogja, hogyha újonnan készített(Onnan hogy nincs neve a seregnek) a sereg és kilépésnél automatikusan felajánlja a lementését. Ha egy már alapból létrehozott seregből lépünk ki akkor az automatikusan mentésre kerül.

Static Public Member Functions inherited from ui

- static void **clearScreen** ()

  Letisztítja a képernyőt. Letörli a képernyőn kiírt adatokat, hogy az újakat "tiszta lapra" lehessen kiírni. A tagfüggvény van arra is kezelve ha windows-ról van futtatva vagy linuxról.
- static void delayMilliseconds (unsigned int ms)

  Függvény adott milliszekundum-nyi késleltetésre. Hasznos arra, hogy egy sakk meccs ne egyből fusson le, hanem mindig várjon a gép lépésenként, hogy legyen ideje felfognia a felhasználónak. A tagfüggvény van arra is kezelve ha windows-ról van futtatva vagy linuxról.

4.23.1 Detailed Description

uiEditor osztály, ami az editor megjelenítéséért felelős.

4.23.2 Constructor & Destructor Documentation

4.23.2.1 uiEditor()

```
uiEditor::uiEditor (
            Editor * editor )
```

Konstruktor editorrel paraméterként

4.23.2.2 ~uiEditor()

```
uiEditor::~uiEditor ( )
```

Destruktor

4.23.3 Member Function Documentation

4.23.3.1 display()

```
void uiEditor::display ( )  [override], [virtual]
```

A szerkesztő megjelenítőfüggvénye. Ez 2 másik függvényt foglal magába.

- clearScreen() : letisztítja az előző táblát a képernyőről
- renderTable() : visszarajzolja a táblát, de már az új állapotában.

Implements ui.

4.23.3.2 handleInput()

```
bool uiEditor::handleInput ( )  [override], [virtual]
```

A szerkesztőnek az inputkezelésével foglalkozó függvény.

- Mivel a menüben is megszokott volt, hogy a [0].-ás gomb az a kilépést szolgálja, itt is ha a felhasználó 0-át ad be inputnak akkor kilép az szerkesztőből.
- Egyébként meg, ha nem 0-át ad a felhasználó mint input, akkor az első betű egy létező bábutípusnak a betűjele kell, hogy legyen, vagy nem fog hozzáadódni a táblához.
- A második-harmadik karaktert pedig átkonvertálja int-re, hogy tudjon vele dolgozni a Bábuhozzáadásnál
- Ha a felhasználó bábubetűjel helyett 'D'-t ad meg, akkor az azon a koordinátán lévő bábut próbálja törölni a fv.
- Ha pedig a felhasználó azt írja be, hogy "delete" akkor törli az éppen szerkesztett sereget a lemenett seregek közül.

  Returns

  Ebben a visszatérési értéke nincsen felhasználva.

Implements ui.

4.23.3.3 idle()

```
void uiEditor::idle ( )  [override], [virtual]
```

A szerkesztő inaktív állapota lényegében.(várja a választ a felektől egymás fele) Felhasználó és program közötti kommunikáció a szerkesztőben amik kilépésre nem kap jelet a program.

Implements ui.

4.23.3.4 Run()

```
void uiEditor::Run (
            Editor * editorptr )  [static]
```

Futtatja a paraméterként kapott szerkesztőt Létrehoz neki egy uiEditor-t és azt berakja idle() pozícióba. A fv. tudni fogja, hogyha újonnan készített(Onnan hogy nincs neve a seregnek) a sereg és kilépésnél automatikusan felajánlja a lementését. Ha egy már alapból létrehozott seregből lépünk ki akkor az automatikusan mentésre kerül.

Parameters

editorptr	Az editorre mutató pointer

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

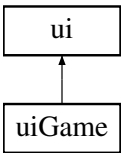
- uiEditor.h
- uiEditor.cpp

4.24 uiGame Class Reference

uiGame osztály, ami a játék megjelenítéséért felelős.

`#include <uiGame.h>`

Inheritance diagram for uiGame:



Public Member Functions

- uiGame (Game *game)
- ~uiGame ()
- void display () override

  A játék megjelenítőfüggvénye. Tartalmaz két alfüggvényt:
- bool handleInput () override

  A játék inputkezelésével foglalkozó függvény.
- void **renderTable** ()

  A játéktábla lerendereléséért felelős függvény. Ez már a teljes 8X8-as táblát meg fogja jeleníteni mindkét csapattal rajta.
- void idle () override

  A játék inaktív állapotát kezelő függvény. Amikor a 2 irányú kommunikáció megy(Itt például a uiGame-ban nincsen felhasználói input, a gép magának adja az inputokat és arra válaszol a megjelenítéssel)
- void **endScreen** ()

  A játék végét jelző képernyőt megjelenítő függvény. Kiírja a játék végén megjelenő üzenetet a képernyőre.(Döntetlen,Csapat1 nyert, Csapat2 nyert)

Public Member Functions inherited from ui

- ui ()

Static Public Member Functions

- static void Run (Game *gameptr)

  Futtatja a paraméterként kapott játékot. Létrehoz neki egy uiGame objektumot és azt berakja idle() pozícióba.

Static Public Member Functions inherited from ui

- static void **clearScreen** ()

  Letisztítja a képernyőt. Letörli a képernyőn kiírt adatokat, hogy az újakat "tiszta lapra" lehessen kiírni. A tagfüggvény van arra is kezelve ha windows-ról van futtatva vagy linuxról.
- static void delayMilliseconds (unsigned int ms)

  Függvény adott milliszekundum-nyi késleltetésre. Hasznos arra, hogy egy sakk meccs ne egyből fusson le, hanem mindig várjon a gép lépésenként, hogy legyen ideje felfognia a felhasználónak. A tagfüggvény van arra is kezelve ha windows-ról van futtatva vagy linuxról.

4.24.1 Detailed Description

uiGame osztály, ami a játék megjelenítéséért felelős.

4.24.2 Constructor & Destructor Documentation

4.24.2.1 uiGame()

```
uiGame::uiGame (
            Game * game )
```

Konstruktor játékobjektummal paraméterként

4.24.2.2 ~uiGame()

```
uiGame::~uiGame ( )
```

Destruktor

4.24.3 Member Function Documentation

4.24.3.1 display()

```
void uiGame::display ( )  [override], [virtual]
```

A játék megjelenítőfüggvénye. Tartalmaz két alfüggvényt:

- clearScreen() : letisztítja az előző táblát a képernyőről
- renderTable() : visszarajzolja a táblát, de már az új állapotában.

Implements ui.

4.24.3.2 handleInput()

```
bool uiGame::handleInput ( )  [override], [virtual]
```

A játék inputkezelésével foglalkozó függvény.

- A felhasználó által bevitt parancsokat értelmezi, és ezek alapján módosítja a játék állapotát.
- Lekezeli a játék befejezését is, ha a játék véget ér, akkor átirányít a végeképernyőre.

Implements ui.

4.24.3.3 idle()

```
void uiGame::idle ( )  [override], [virtual]
```

A játék inaktív állapotát kezelő függvény. Amikor a 2 irányú kommunikáció megy(Itt például a uiGame-ban nincsen felhasználói input, a gép magának adja az inputokat és arra válaszol a megjelenítéssel)

Implements ui.

4.24.3.4 Run()

```
void uiGame::Run (
            Game * gameptr )  [static]
```

Futtatja a paraméterként kapott játékot. Létrehoz neki egy uiGame objektumot és azt berakja idle() pozícióba.

Parameters

gameptr	A játékra mutató pointer

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

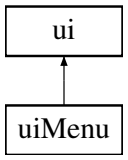
- uiGame.h
- uiGame.cpp

4.25 uiMenu Class Reference

uiMenu osztály, ami a menü megjelenítéséért felelős.

`#include <uiMenu.h>`

Inheritance diagram for uiMenu:



Public Member Functions

- uiMenu (Menu *menu)
- ~uiMenu ()
- void display () override

  A menü megjelenítőfüggvénye. Kirajzolja a menüt a képernyőre.
- bool handleInput () override

  A menü inputkezelésével foglalkozó függvény. A felhasználó által bevitt parancsokat értelmezi, és ezek alapján választ gombot a menüben.
- void idle () override

  A menü inaktív állapotát kezelő függvény. Amikor a menü vár arra, hogy válasszunk egy gombot.
- void **refreshingidle** ()

  Ez az idle() annyiban különbözik az előzőtől, hogy ez megnézi a handleInput() visszatérési értékét, és ha helyesen futott le a függvény utána kilép majd a menüből.

Public Member Functions inherited from ui

- ui ()

Static Public Member Functions

- static void Run (Menu *menu)

  Futtatja a paraméterként kapott menüt. Létrehoz neki egy uiMenu objektumot és azt berakja idle() pozícióba.
- static void refreshingRun (Menu *menuPtr)

  Futtatja a paraméterként kapott menüt. Létrehoz neki egy uiMenu objektumot és azt berakja refreshingidle() pozícióba.

Static Public Member Functions inherited from ui

- static void **clearScreen** ()

  Letisztítja a képernyőt. Letörli a képernyőn kiírt adatokat, hogy az újakat "tiszta lapra" lehessen kiírni. A tagfüggvény van arra is kezelve ha windows-ról van futtatva vagy linuxról.
- static void delayMilliseconds (unsigned int ms)

  Függvény adott milliszekundum-nyi késleltetésre. Hasznos arra, hogy egy sakk meccs ne egyből fusson le, hanem mindig várjon a gép lépésenként, hogy legyen ideje felfognia a felhasználónak. A tagfüggvény van arra is kezelve ha windows-ról van futtatva vagy linuxról.

4.25.1 Detailed Description

uiMenu osztály, ami a menü megjelenítéséért felelős.

4.25.2 Constructor & Destructor Documentation

4.25.2.1 uiMenu()

```
uiMenu::uiMenu (
            Menu * menu )
```

Konstruktor menüobjektummal paraméterként

4.25.2.2 ~uiMenu()

```
uiMenu::~uiMenu ( )
```

Destruktor

4.25.3 Member Function Documentation

4.25.3.1 display()

```
void uiMenu::display ( )  [override], [virtual]
```

A menü megjelenítőfüggvénye. Kirajzolja a menüt a képernyőre.

Implements ui.

4.25.3.2 handleInput()

```
bool uiMenu::handleInput ( )  [override], [virtual]
```

A menü inputkezelésével foglalkozó függvény. A felhasználó által bevitt parancsokat értelmezi, és ezek alapján választ gombot a menüben.

Returns

true: sikeresen megnyomott gomb

false:kilépés a függvényből, vagy hibás bemenet.

Implements ui.

4.25.3.3 idle()

```
void uiMenu::idle ( )  [override], [virtual]
```

A menü inaktív állapotát kezelő függvény. Amikor a menü vár arra, hogy válasszunk egy gombot.

Implements ui.

4.25.3.4 refreshingRun()

```
void uiMenu::refreshingRun (
            Menu * menuPtr )  [static]
```

Futtatja a paraméterként kapott menüt. Létrehoz neki egy uiMenu objektumot és azt berakja refreshingidle() pozícióba.

Parameters

menuPtr	A menüre mutató pointer

4.25.3.5 Run()

```
void uiMenu::Run (
            Menu * menu )  [static]
```

Futtatja a paraméterként kapott menüt. Létrehoz neki egy uiMenu objektumot és azt berakja idle() pozícióba.

Parameters

menuPtr	A menüre mutató pointer

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

- uiMenu.h
- uiMenu.cpp

Chapter 5

File Documentation

5.1 Army.h File Reference

Az Army osztályt tartalmazó header.

```
#include "Piece.h"
#include <cstring>
```

Classes

- class Army

  Sereg class és annak a függvényei.

5.1.1 Detailed Description

Az Army osztályt tartalmazó header.

5.2 Army.h

Go to the documentation of this file.

```
00001
00006 #ifndef NHF_REFORMED_ARMY_H
00007 #define NHF_REFORMED_ARMY_H
00008 #include "Piece.h"
00009 #include <cstring>
00010
00014 class Army {
00015 private:
00016     char nameofArmy[36];
00017     int sizeofArmy;
00018     List<Piece> pieces;
00020 public:
00025     Army();
00026
00031     ~Army();
00032
00037     const char* getnameofArmy()const;
00038
00043     void setnameofArmy(const char* name);
00044
```

```
00049     int getsizeofArmy()const;
00050
00055     void setsizeofArmy(int size);
00056
00060     void incrementsizeofArmy();
00061
00067     void addPiece(Piece& newPiece);
00068
00075     void deletePiece(int coordX, int coordY);
00076
00084     Piece* getPiece(int coordX, int coordY);
00085
00093     bool partOfArmy(Piece* piece);
00094
00099     Army(const Army& army);
00100
00106     Army& operator=(const Army& army);
00107
00114     Piece* getPiece(size_t idx);
00115
00121     static void copyArmy(Army* source, Army* destination);
00122
00129     static void mirrorArmy(Army* army);
00130 };
00131 #endif //NHF_REFORMED_ARMY_H
```

5.3 Bishop.h File Reference

Bishop osztályt tartalmazó header.

```
#include "Piece.h"
```

Classes

- class Bishop

  Bishop osztály, ami a futót reprezentálja a sakkjátékban.

5.3.1 Detailed Description

Bishop osztályt tartalmazó header.

5.4 Bishop.h

Go to the documentation of this file.

```
00001
00006 #ifndef NHF_REFORMED_BISHOP_H
00007 #define NHF_REFORMED_BISHOP_H
00008 #include "Piece.h"
00009
00013 class Bishop:public Piece{
00014 public:
00015
00021     Bishop(int x,int y);
00022
00028     void calculateMoves(Game* game) override;
00029 };
00030 #endif //NHF_REFORMED_BISHOP_H
```

5.5 button.h File Reference

A Button osztályt tartalmazó header.

```
#include "Army.h"
#include "ButtonFunctionHandler.h"
```

Classes

- class Button

  A gomb class. Ezekből épülnek fel a menük. Minden gombnak van egy id-je. Ezeket az id-ket a gombot tartalmazó Menü fogja kiosztani a gomboknak amilyen sorrendben jönnek. Minden gomb tartalmaz egy funkciót ami futtathat menüt/szerkesztőt/játékot.

5.5.1 Detailed Description

A Button osztályt tartalmazó header.

5.6 button.h

Go to the documentation of this file.

```
00001
00006 #ifndef NHF_REFORMED_BUTTON_H
00007 #define NHF_REFORMED_BUTTON_H
00008 #include "Army.h"
00009 #include "ButtonFunctionHandler.h"
00010
00016 class Button{
00017 private:
00018     size_t id;
00019     char* name;
00020     ButtonFunctionHandler buttonFunction;
00022 public:
00026     Button();
00027
00034     Button(const char* name,size_t id,void (*function)());
00035
00043     Button(const char* name,size_t id,void (*functionArmy)(Army*),Army* armyPtr);
00044
00053     Button(const char* name,size_t id,void (*functionArmy)(Army* first,Army* second),Army*
      armyPtr1,Army* armyPtr2);
00054
00059     ~Button();
00060
00064     Button(const Button& other);
00065
00069     Button& operator=(const Button& other);
00070
00075     void setId(size_t id);
00076
00081     size_t getId()const;
00082
00087     const char* getName()const;
00088
00093     ButtonFunctionHandler getFunction()const;
00094 };
00095 #endif //NHF_REFORMED_BUTTON_H
```

5.7 ButtonFunctionHandler.h File Reference

A ButtonFunctionHandler osztályt tartalmazó header.

Classes

- class ButtonFunctionHandler

  Hozzákapcsolja a function-pointereket a gombokhoz. Emellett lehetővé teszi, hogy több fajta függvényeket is tudjon futtatni egy gomb, akár olyanokat is amiknek más paramétereik vannak.

5.7.1 Detailed Description

A ButtonFunctionHandler osztályt tartalmazó header.

5.8 ButtonFunctionHandler.h

Go to the documentation of this file.

```
00001
00006 #ifndef NHF_REFORMED_BUTTONFUNCTIONHANDLER_H
00007 #define NHF_REFORMED_BUTTONFUNCTIONHANDLER_H
00008
00009 class Army;
00010
00015 class ButtonFunctionHandler{
00016 private:
00017     void (*fun)();
00018     void (*funArmy)(Army *army);
00019     void (*funArmyFor2)(Army* army1, Army* army2);
00021     Army* regArmy1;
00022     Army* regArmy2;
00024 public:
00028     ~ButtonFunctionHandler(){}
00029
00033     ButtonFunctionHandler(): fun(nullptr), funArmy(nullptr), funArmyFor2(nullptr) , regArmy1(nullptr),
      regArmy2(nullptr){}
00034
00039     ButtonFunctionHandler(void (*func)()): fun(func), funArmy(nullptr), funArmyFor2(nullptr) ,
      regArmy1(nullptr), regArmy2(nullptr){}
00040
00046     ButtonFunctionHandler(void (*funcArmy)(Army*), Army* armyPtr): fun(nullptr), funArmy(funcArmy),
      funArmyFor2(nullptr) , regArmy1(armyPtr), regArmy2(nullptr){}
00047
00054     ButtonFunctionHandler(void (*funcArmy)(Army*, Army*), Army* armyPtr1, Army* armyPtr2):
      fun(nullptr), funArmy(nullptr), funArmyFor2(funcArmy), regArmy1(armyPtr1), regArmy2(armyPtr2){}
00055
00059     void execute();
00060 };
00061 #endif //NHF_REFORMED_BUTTONFUNCTIONHANDLER_H
```

5.9 buttonfunctions.h File Reference

Ez a header tartalmazza a függvénypointereket, amik aktiválódnak amikor egy gombot nyomunk le egy menüben.

```
#include "Army.h"
```

Classes

- class ButtonFunctions

  Függvénypointerek, amik aktiválódnak ha lenyomják az őket tartalmazó gombot.

5.9.1 Detailed Description

Ez a header tartalmazza a függvénypointereket, amik aktiválódnak amikor egy gombot nyomunk le egy menüben.

5.10 buttonfunctions.h

Go to the documentation of this file.

```
00001
00006 #ifndef NHF_REFORMED_BUTTONFUNCTIONS_H
00007 #define NHF_REFORMED_BUTTONFUNCTIONS_H
00008 #include "Army.h"
00009
00010
00014 class ButtonFunctions{
00015 public:
00016     static void MainMenu();
00025     static void Play();
00026
00032     static void NewGame();
00033
00038     static void ArmyMenu();
00039
00043     static void CreateArmy();
00044
00048     static void EditArmy(Army*);
00049
00054     static void ChooseArmy(Army*); //a paraméterként megkapott paraméter-be tölti majd be az armyt
      amit választ
00055
00061     static void PlayMatch(Army* reg1, Army* reg2);
00062 };
00063 #endif //NHF_REFORMED_BUTTONFUNCTIONS_H
```

5.11 Computer.h File Reference

A Computer osztályt tartalmazó header.

```
#include "Team.h"
```

Classes

- class Computer

  Computer class az, aki kiszámolja a lehetséges lépéseket. Megállapítja, hogy melyik csapat jön a játékban, majd kigyűjti a csapat lépéseit.

5.11.1 Detailed Description

A Computer osztályt tartalmazó header.

5.12 Computer.h

Go to the documentation of this file.

```
00001
00006 #ifndef NHF_REFORMED_COMPUTER_H
00007 #define NHF_REFORMED_COMPUTER_H
00008 #include "Team.h"
00009
00014 class Computer {
00015 public:
00019     Computer();
00020
00029     void calculateMoves(Game* game);
00030
00037     Move* decideMove(Game* game);
00038 };
00039 #endif //NHF_REFORMED_COMPUTER_H
```

5.13 Editor.h File Reference

Az Editor osztályt tartalmazó header.

```
#include "Army.h"
#include "Filemanagement.h"
#include <sstream>
#include <string>
```

Classes

- class Editor

  A seregek létrehozásáért és szerkesztéséért felelős osztály.

5.13.1 Detailed Description

Az Editor osztályt tartalmazó header.

5.14 Editor.h

Go to the documentation of this file.

```
00001
00006 #ifndef NHF_REFORMED_EDITOR_H
00007 #define NHF_REFORMED_EDITOR_H
00008 #include "Army.h"
00009 #include "Filemanagement.h"
00010 #include <sstream>
00011 #include <string>
00012
00016 class Editor {
00017
00018 private:
00019     Army* army;
00020     bool exit;
00021     bool toDelete;
00023 public:
00028     Editor();
00029
00034     ~Editor();
00035
00041     Editor(Army* army);
```

```
00042
00047     Army* getArmy() const;
00048
00055     Piece* searchFor(int coordX, int coordY);
00056
00060     void updateExit();
00061
00066     bool getExit() const;
00067
00071     void updateDelete();
00072
00077     bool getDelete() const;
00078
00083     void saveArmy();
00084
00089     void editArmy();
00090
00095     void deleteArmy();
00096 };
00097 #endif //NHF_REFORMED_EDITOR_H
```

5.15 Error.h File Reference

Az error osztály deklarációit tartalmazza.

```
#include <stdexcept>
```

Classes

- class Error

  Saját error osztály. Az osztály az std::exception-ből származik és külön hibaüzenetet produkál.

5.15.1 Detailed Description

Az error osztály deklarációit tartalmazza.

5.16 Error.h

Go to the documentation of this file.

```
00001
00006 #ifndef NHF_REFORMED_ERROR_H
00007 #define NHF_REFORMED_ERROR_H
00008 #include <stdexcept>
00009
00010
00015 class Error:public std::exception {
00016
00017 private:
00018     std::string message;
00020 public:
00025     Error(std::string& message);
00026
00027
00032     const char* what() const noexcept;
00033 };
00034 #endif //NHF_REFORMED_ERROR_H
```

5.17 Filemanagement.h File Reference

A fájlkezeléssel foglalkozó osztály és annak deklarációi.

```
#include <fstream>
#include "List.hpp"
#include "Army.h"
#include "Piece.h"
#include <vector>
#include <string>
#include <sstream>
```

Classes

- class Filemanagement

  Fájlkezelő osztály. Legfőkébb az elkészített seregek tárolását valósítja meg az armies.txt-ben.

5.17.1 Detailed Description

A fájlkezeléssel foglalkozó osztály és annak deklarációi.

5.18 Filemanagement.h

Go to the documentation of this file.

```
00001
00007 #ifndef NHF_REFORMED_FILEMANAGEMENT_H
00008 #define NHF_REFORMED_FILEMANAGEMENT_H
00009 #include <fstream>
00010 #include "List.hpp"
00011 #include "Army.h"
00012 #include "Piece.h"
00013 #include <vector>
00014 #include <string>
00015 #include <sstream>
00016
00021 class Filemanagement{
00022
00023 public:
00024
00031     static List<Army> ListofArmies(const char* filename);
00032
00039     static void AppendArmy(Army* army,const char* filename);
00040
00057     static void EditArmy(Army* army,const char* filename);
00058
00071     static void DeleteArmy(Army* army,const char* filename);
00072
00073 private:
00074
00082     static bool ifFileNonExistentCreate(const char* filename);
00083
00091     static void readTillLine(std::ifstream& file, std::vector<std::string>& lines,const std::string&
      boundary="");
00092
00099     static void skipLines(std::ifstream& file,int numberofLines);
00100
00107     static void writeLines(std::ofstream& file,std::vector<std::string>& lines);
00108
00115     static void writeArmy(std::ofstream& file,Army* army);
00116
00128     static void overwriteline(const char* filename,std::string newLine,int lineNumber);
00129
00136     static void adjustStringNumber(std::string& string,bool increment);
00137 };
00138 #endif //NHF_REFORMED_FILEMANAGEMENT_H
```

5.19 Game.h File Reference

A GameResult enumot és Game osztályt tartalmazó header.

```
#include "Team.h"
#include "Army.h"
#include "Computer.h"
```

Classes

- class Game

  A játék class. Ez van használatban amikor 2 csapat játszik.

Enumerations

- enum GameResult { TEAM1_WIN , TEAM2_WIN , DRAW }

  A játékeredmény lehetséges értékei A játékban alapból DRAW-ként lesz inicializálva.

5.19.1 Detailed Description

A GameResult enumot és Game osztályt tartalmazó header.

5.19.2 Enumeration Type Documentation

5.19.2.1 GameResult

```
enum GameResult
```

A játékeredmény lehetséges értékei A játékban alapból DRAW-ként lesz inicializálva.

Enumerator

TEAM1_WIN	Első csapat nyertes
TEAM2_WIN	Második csapat nyertes
DRAW	Döntetlen

5.20 Game.h

Go to the documentation of this file.

```
00001
00006 #ifndef NHF_REFORMED_GAME_H
00007 #define NHF_REFORMED_GAME_H
00008 #include "Team.h"
00009 #include "Army.h"
00010 #include "Computer.h"
```

```
00011
00016 enum GameResult {
00017     TEAM1_WIN,
00018     TEAM2_WIN,
00019     DRAW
00020 };
00021
00022
00026 class Game {
00027 private:
00028     Team* team[2];
00029     Computer computer;
00030     bool WhiteTurn;
00031     bool endOfGame;
00032     GameResult result;
00034 public:
00038     Game();
00039
00044     ~Game();
00045
00052     Game(Army* white,Army* black);
00053
00060     bool isWhiteTurn()const;
00061
00068     bool getEnd()const;
00069
00074     GameResult getResult()const;
00075
00080     void updateEnd();
00081
00089     Piece* searchFor(int x, int y);
00090
00097     Team* getTeam(size_t idx);
00098
00104     TeamColor getColorOfPiece(Piece* piece);
00105
00114     bool occupied(int x,int y);
00115
00125     void makeMove();
00126
00132     void playRound();
00133
00142     void checkIfOver();
00143
00148     void clearMovesBuffer();
00149 };
00150 #endif //NHF_REFORMED_GAME_H
```

5.21 Horse.h File Reference

Horse osztályt tartalmazó header.

```
#include "Piece.h"
```

Classes

- class Horse

5.21.1 Detailed Description

Horse osztályt tartalmazó header.

5.22 Horse.h

Go to the documentation of this file.

```
00001 #ifndef NHF_REFORMED_HORSE_H
00002 #define NHF_REFORMED_HORSE_H
00003 #include "Piece.h"
00004
00005
00010 class Horse:public Piece {
00011 public:
00012
00018     Horse(int x,int y);
00019
00025     void calculateMoves(Game* game) override;
00026 };
00027 #endif //NHF_REFORMED_HORSE_H
```

5.23 King.h File Reference

King osztályt tartalmazó header.

```
#include "Piece.h"
```

Classes

- class King

  King osztály, ami a királyt reprezentálja a sakkjátékban. Ebből a bábuból minden csapatnak kell, hogy legyen különben automatikusan veszt. Ez a játék feltétele. Ebből lehet több egy csapatban.

5.23.1 Detailed Description

King osztályt tartalmazó header.

5.24 King.h

Go to the documentation of this file.

```
00001
00006 #ifndef NHF_REFORMED_KING_H
00007 #define NHF_REFORMED_KING_H
00008 #include "Piece.h"
00009
00014 class King:public Piece {
00015 public:
00016
00022     King(int x,int y);
00023
00029     void calculateMoves(Game* game) override;
00030 };
00031 #endif //NHF_REFORMED_KING_H
```

5.25 List.hpp File Reference

A generikus listát tartalmazó .hpp.

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

Classes

- struct Node< T >
- class List< T >

5.25.1 Detailed Description

A generikus listát tartalmazó .hpp.

5.26 List.hpp

Go to the documentation of this file.

```
00001
00005 #include <iostream>
00006 #include "memtrace.h"
00007 #ifndef NHF_REFORMED_LIST_HPP
00008 #define NHF_REFORMED_LIST_HPP
00009
00010 template <class T>
00011 struct Node{
00012     T *data;
00013     Node<T>* previous;
00014     Node<T>* next;
00016     Node(T* newData):data(newData),previous(nullptr),next(nullptr){}
00017     ~Node(){delete data;}
00018     T* getData(){
00019         return data;
00020     }
00021     T* release(){
00022         T* ptr = data;
00023         data = nullptr;
00024         return ptr;
00025     }
00026 };
00027
00028 template <class T>
00029 class List {
00030 private:
00031     Node<T>* head;
00032     Node<T>* tail;
00033     size_t size;
00034 public:
00035
00039     List() : head(nullptr), tail(nullptr), size(0) {}
00040
00044     ~List(){
00045         for(Node<T>* i=head;i!=nullptr;i=i->next){
00046             if(i->previous!=nullptr){delete i->previous;}
00047             if(i==tail){delete i;break;}
00048         }
00049     }
00050
00054     void addtoList(T* newData){
00055         Node<T>* newNode= new Node<T>(newData);
00056         if(this->head==nullptr){
00057             this->head = newNode;
00058             this->tail = newNode;
00059             size++;
00060             return;
00061         }
00062         this->tail->next=newNode;
00063         newNode->previous=this->tail;
00064         this->tail=newNode;
00065         size++;
00066     }
00067
00071     void deletefromList(T* todelete){
00072         if(this->head==nullptr){return;}
00073         for(Node<T>* i=head;i!=nullptr;i=i->next){
00074             if(i->data==todelete){
00075                 if(size==1){
00076                     head=nullptr;
00077                     tail=nullptr;
00078                 }
00079                 else if(i==tail){
```

```
00080                   i->previous->next=nullptr;
00081                   tail=i->previous;
00082
00083               }
00084               else if(i==head){
00085                   i->next->previous=nullptr;
00086                   head=i->next;
00087               }
00088               else{
00089                   i->previous->next=i->next;
00090                   i->next->previous=i->previous;
00091               }
00092               delete i;
00093               size--;
00094               return;
00095           }
00096       }
00097   }
00098
00102   List(const List& list){
00103       if(list.head==nullptr){
00104           this->head=nullptr;
00105           this->tail=nullptr;
00106           this->size=0;
00107           return;
00108       }
00109       this->size=0;
00110       for(Node<T>* i=list.head;i!=nullptr;i=i->next){
00111           this->addtoList(i->data);
00112       }
00113   }
00114
00118   List& operator=(const List& list){
00119       if(this != &list) {
00120           if (this->head != nullptr) {
00121               for (Node<T> *i = head; i != nullptr; i = i->next) {
00122                   if (i != head) { delete i->previous; }
00123                   if (i == tail) {
00124                       delete i;
00125                       head = nullptr;
00126                       tail = nullptr;
00127                       break;
00128                   }
00129               }
00130           }
00131           this->size = 0;
00132           for (Node<T> *i=list.head; i != nullptr; i = i->next) {
00133               this->addtoList((i->data));
00134           }
00135       }
00136       return *this;
00137   }
00138
00142   int getSize()const{
00143       return size;
00144   }
00145
00149   T* operator[](size_t index)const{
00150       if(index>=size){
00151           return nullptr;
00152       }
00153       Node<T>* temp=head;
00154       for(size_t i=0;i!=index;i++){
00155           temp=temp->next;
00156       }
00157       return temp->getData();
00158   }
00159
00163   void clear(){
00164       if(head==nullptr){return;}
00165       while(head!=nullptr){
00166           Node<T>* temp = head;
00167           head = head->next;
00168           delete temp;
00169       }
00170       tail = nullptr;
00171       size = 0;
00172   }
00173
00177   void consumeList(List& consumed){
00178       if(consumed.head==nullptr){return;}
00179       for(Node<T>* i=consumed.head;i!=nullptr;i=i->next){
00180           this->addtoList(i->release());
00181       }
00182       consumed.clear();
00183   }
00184
```

```
00188     T* Maximum(){
00189         T* maximum = nullptr;
00190         if(this->size==0){return maximum;}
00191         maximum = this->head->getData();
00192         for(Node<T>* i=head->next;i!=nullptr;i=i->next){
00193             if( (*(i->getData()))>(*maximum) ){maximum = i->getData();}
00194         }
00195         return maximum;
00196     }
00197 };
00198 #endif //NHF_REFORMED_LIST_HPP
```

5.27 memtrace.h

```
00001 #ifndef NHF_REFORMED_MEMTRACE_H
00002 #define NHF_REFORMED_MEMTRACE_H
00003
00004 #endif //NHF_REFORMED_MEMTRACE_H
```

5.28 menu.h File Reference

Ez a header tartalmazza a Menu osztályt, aminek köszönhetően lehet bejárni a programot.

```
#include "button.h"
#include <iostream>
#include <cstddef>
```

Classes

- class Menu

  Menü class, gombokat tárol.

5.28.1 Detailed Description

Ez a header tartalmazza a Menu osztályt, aminek köszönhetően lehet bejárni a programot.

5.29 menu.h

Go to the documentation of this file.

```
00001
00006 #ifndef NHF_REFORMED_MENU_H
00007 #define NHF_REFORMED_MENU_H
00008 #include "button.h"
00009 #include <iostream>
00010 #include <cstddef>
00011
00012
00016 class Menu{
00017 private:
00018     size_t idCounter;
00019     Button* buttonArray;
00020     bool exitRequested;
00021 public:
00025     Menu();
00026
00031     ~Menu();
00032
00036     void incrementCounter();
```

```
00037
00041     void updateExit();
00042
00051     void addButton(const Button& extra);
00052
00058     Button* getButton(int idx)const;
00059
00066     bool getExit()const;
00067
00073     size_t getIdCounter()const;
00074 };
00075
00076 #endif //NHF_REFORMED_MENU_H
```

5.30 Move.h File Reference

A Move osztályt tartalmazó header.

Classes

- class Move

  A Move class. Ezt tárolják el a bábuk, emellett csapatok listákban.

5.30.1 Detailed Description

A Move osztályt tartalmazó header.

5.31 Move.h

Go to the documentation of this file.

```
00001
00006 #ifndef NHF_REFORMED_MOVE_H
00007 #define NHF_REFORMED_MOVE_H
00008
00009 class Piece;
00010
00011
00015 class Move {
00016 private:
00017     Piece* originPiece;
00018     int destinationX, destinationY;
00019     char destinationPieceName;
00020     int weight;
00022 public:
00023
00032     Move(Piece* originPiece, int coordX, int coordY, char destinationPieceName=0);
00033
00037     ~Move();
00038
00043     void calculateWeight();
00044
00052     bool operator>(const Move& otherMove) const;
00053
00059     Piece* getPiece()const;
00060
00065     int getCoordX()const;
00066
00071     int getCoordY()const;
00072
00077     int getWeight()const;
00078 };
00079 #endif //NHF_REFORMED_MOVE_H
```

5.32 Pawn.h File Reference

Pawn osztályt tartalmazó header.

```
#include "Piece.h"
```

Classes

- class Pawn

  Pawn osztály, ami a gyalogot reprezentálja a sakkjátékban.

5.32.1 Detailed Description

Pawn osztályt tartalmazó header.

5.33 Pawn.h

Go to the documentation of this file.

```
00001
00006 #ifndef NHF_REFORMED_PAWN_H
00007 #define NHF_REFORMED_PAWN_H
00008 #include "Piece.h"
00009
00013 class Pawn:public Piece {
00014 public:
00020     Pawn(int x,int y);
00021
00027     void calculateMoves(Game* game) override;
00028 };
00029 #endif //NHF_REFORMED_PAWN_H
```

5.34 Piece.h File Reference

Piece osztályt és a lépésüket kiszámoló statikus függvények headerje.

```
#include "List.hpp"
#include "Move.h"
```

Classes

- class Piece

  Bábu osztály. Ez egy absztrakt osztály, amiből az összes sakkbábu osztály öröklődik. Belőlük épülnek fel a seregek.

5.34.1 Detailed Description

Piece osztályt és a lépésüket kiszámoló statikus függvények headerje.

5.35 Piece.h

Go to the documentation of this file.

```
00001
00006 #ifndef NHF_REFORMED_PIECE_H
00007 #define NHF_REFORMED_PIECE_H
00008 #include "List.hpp"
00009 #include "Move.h"
00010
00011 class Game;
00012
00013
00019 class Piece {
00020 private:
00021     int y,x;
00022     char name;
00024 protected:
00025     List<Move> piece_moves;
00027 public:
00034     Piece(char name, int coordX=0, int coordY=0);
00035
00040     virtual ~Piece(){}
00041
00046     int getCoordY()const;
00047
00052     int getCoordX()const;
00053
00058     char getName()const;
00059
00064     List<Move>& getMoves();
00065
00070     void setCoordY(int newY);
00071
00076     void setCoordX(int newX);
00077
00084     bool operator==(const Piece& otherPiece)const;
00085
00090     void toLowercase();
00091
00097     void virtual calculateMoves(Game* game)=0;
00098
00106     void addMove(char destinationPieceName, int coordX, int coordY);
00107
00116     static Piece* createPiece(char name,int x,int y);
00117
00129     static bool checkAndAddMove(Game* game, Piece* originPiece, int posX, int posY);
00130
00137     static void upwards(Piece* originPiece, Game* game);//Észak,dél,kelet,nyugat
00138
00145     static void downwards(Piece* originPiece, Game* game);
00146
00153     static void rightwards(Piece* originPiece, Game* game);
00154
00161     static void leftwards(Piece* originPiece, Game* game);
00162
00169     static void diagonalUpRight(Piece* originPiece, Game* game);//ÉK,ÉNY,DK,DNY
00170
00177     static void diagonalUpLeft(Piece* originPiece, Game* game);
00178
00185     static void diagonalDownRight(Piece* originPiece, Game* game);
00186
00193     static void diagonalDownLeft(Piece* originPiece, Game* game);
00194
00201     static void orthogonal(Piece* originPiece, Game* game);
00202
00209     static void diagonal(Piece* originPiece, Game* game);
00210
00220     static void pawnMove(Piece* originPiece, Game* game);
00221
00228     static void horseMove(Piece* originPiece, Game* game);
00229
00236     static void kingMove(Piece* originPiece, Game* game);
00237 };
00238 #endif //NHF_REFORMED_PIECE_H
```

5.36 Queen.h File Reference

Queen osztályt tartalmazó header.

```
#include "Piece.h"
```

Classes

- class Queen

  Queen osztály, ami a vezért reprezentálja a sakkjátékban.

5.36.1 Detailed Description

Queen osztályt tartalmazó header.

5.37 Queen.h

Go to the documentation of this file.

```
00001
00006 #ifndef NHF_REFORMED_QUEEN_H
00007 #define NHF_REFORMED_QUEEN_H
00008 #include "Piece.h"
00009
00013 class Queen:public Piece {
00014 public:
00020     Queen(int x,int y);
00021
00027     void calculateMoves(Game* game) override;
00028 };
00029 #endif //NHF_REFORMED_QUEEN_H
```

5.38 Rook.h File Reference

Rook osztályt tartalmazó header.

```
#include "Piece.h"
```

Classes

- class Rook

  Rook osztály, ami a bástyát reprezentálja a sakkjátékban.

5.38.1 Detailed Description

Rook osztályt tartalmazó header.

5.39 Rook.h

Go to the documentation of this file.

```
00001
00006 #ifndef NHF_REFORMED_ROOK_H
00007 #define NHF_REFORMED_ROOK_H
00008 #include "Piece.h"
00009
00013 class Rook:public Piece {
00014 public:
00020     Rook(int x,int y);
00021
00027     void calculateMoves(Game* game) override;
00028 };
00029 #endif //NHF_REFORMED_ROOK_H
```

5.40 Team.h File Reference

A Team osztályt és TeamColor-t tartalmazó header.

```
#include "List.hpp"
#include "Piece.h"
#include "Queen.h"
#include "King.h"
#include "Pawn.h"
#include "Horse.h"
#include "Bishop.h"
#include "Rook.h"
#include "Army.h"
```

Classes

- class Team

  A Team class. Ebből jön létre 2 db amikor elindul a játék-szimuláció. Ide vannak kigyűjtve a seregek lépései miután a bábuk összegyűjtötték a lehetséges lépéseiket.

Enumerations

- enum TeamColor { **White** , **Black** }

  A csapatszínek lehetséges értékei.

5.40.1 Detailed Description

A Team osztályt és TeamColor-t tartalmazó header.

5.41 Team.h

Go to the documentation of this file.

```
00001
00006 #ifndef NHF_REFORMED_TEAM_H
00007 #define NHF_REFORMED_TEAM_H
00008 #include "List.hpp"
00009 #include "Piece.h"
00010 #include "Queen.h"
00011 #include "King.h"
00012 #include "Pawn.h"
00013 #include "Horse.h"
00014 #include "Bishop.h"
00015 #include "Rook.h"
00016 #include "Army.h"
00017 #include "Piece.h"
00018
00022 enum TeamColor{White,Black};
00023
00024
00029 class Team {
00030 private:
00031     List<Move> teamMoves;
00032     TeamColor teamColor;
00033     Army* ownArmy;
00034 public:
00035
00043     Team(Army* army,TeamColor color);
00044
```

```
00049     ~Team();
00050
00055     Army* getArmy();
00056
00061     List<Move>& getTeamMoves();
00062
00067     Move* getRandomMove();
00068
00074     int countAmountOfKigns();
00075 };
00076 #endif //NHF_REFORMED_TEAM_H
```

5.42 ui.h File Reference

Ez a header tárolja a ui(user interface) absztrakt osztályt.

Classes

- class ui

  ui absztakt osztály Ebből származik a uiGame, uiEditor, uiMenu. Ezek az osztályok tartják fent a kapcsolatot a felhasználó és számítógép között.

5.42.1 Detailed Description

Ez a header tárolja a ui(user interface) absztrakt osztályt.

5.43 ui.h

Go to the documentation of this file.

```
00001
00006 #ifndef NHF_REFORMED_UI_H
00007 #define NHF_REFORMED_UI_H
00008
00009
00015 class ui {
00016 public:
00017     ui(){}
00023     virtual void display()=0;
00024
00025
00026     virtual bool handleInput()=0;
00032     virtual void idle()=0;
00033
00039     static void clearScreen();
00040
00047     static void delayMilliseconds(unsigned int ms);
00048 };
00049 #endif //NHF_REFORMED_UI_H
```

5.44 uiEditor.h File Reference

uiEditor osztályt tartalmazó header

```
#include "Editor.h"
#include "ui.h"
#include <iostream>
#include "Filemanagement.h"
```

Classes

- class uiEditor

  uiEditor osztály, ami az editor megjelenítéséért felelős.

5.44.1 Detailed Description

uiEditor osztályt tartalmazó header

5.45 uiEditor.h

Go to the documentation of this file.

```
00001
00006 #ifndef NHF_REFORMED_UIEDITOR_H
00007 #define NHF_REFORMED_UIEDITOR_H
00008 #include "Editor.h"
00009 #include "ui.h"
00010 #include <iostream>
00011 #include "Filemanagement.h"
00012
00013
00017 class uiEditor:public ui {
00018 private:
00019     Editor* editor;
00020 public:
00021     uiEditor(Editor* editor);
00022     ~uiEditor();
00030     void display() override;
00031
00041     bool handleInput() override;
00042
00048     void renderTable();
00049
00054     void idle()override;
00055
00060     void saveSequence();
00061
00069     static void Run(Editor* editorptr);
00070 };
00071 #endif //NHF_REFORMED_UIEDITOR_H
```

5.46 uiGame.h File Reference

uiGame osztályt tartalmazó header

```
#include "ui.h"
#include "Game.h"
#include <iostream>
```

Classes

- class uiGame

  uiGame osztály, ami a játék megjelenítéséért felelős.

5.46.1 Detailed Description

uiGame osztályt tartalmazó header

5.47 uiGame.h

Go to the documentation of this file.

```
00001
00006 #ifndef NHF_REFORMED_UIGAME_H
00007 #define NHF_REFORMED_UIGAME_H
00008 #include "ui.h"
00009 #include "Game.h"
00010 #include <iostream>
00011
00015 class uiGame:public ui {
00016 private:
00017     Game* game;
00018 public:
00019     uiGame(Game* game);
00020     ~uiGame();
00028     void display() override;
00029
00035     bool handleInput() override;
00036
00041     void renderTable();
00042
00047     void idle() override;
00048
00053     void endScreen();
00054
00060     static void Run(Game* gameptr);
00061 };
00062 #endif //NHF_REFORMED_UIGAME_H
```

5.48 uiMenu.h File Reference

uiMenu osztályt tartalmazó header

```
#include "menu.h"
#include "ui.h"
```

Classes

- class uiMenu

  uiMenu osztály, ami a menü megjelenítéséért felelős.

5.48.1 Detailed Description

uiMenu osztályt tartalmazó header

5.49 uiMenu.h

Go to the documentation of this file.

```
00001
00006 #ifndef NHF_REFORMED_UIMENU_H
00007 #define NHF_REFORMED_UIMENU_H
00008 #include "menu.h"
00009 #include "ui.h"
00010
00014 class uiMenu:public ui {
00015 private:
00016     Menu* menu;
00017 public:
00018     uiMenu(Menu* menu);
```

```
00019     ~uiMenu();
00025     void display() override;
00026
00033     bool handleInput() override;
00034
00039     void idle() override;
00040
00044     void refreshingidle();
00045
00051     static void Run(Menu* menu);
00052
00058     static void refreshingRun(Menu* menuPtr);
00059 };
00060 #endif //NHF_REFORMED_UIMENU_H
```

Index