Sakk Dokumentáció

**Feladatleírás:**

Tervezzen "demokratikus sakk" modellezésére objektummodellt! Ebben a játékban a táblán a figurák önállóan döntenek, hogy hova lépnek. Minden figura tudja a saját szabályait. A megvalósítandó modellben felváltva választunk egy-egy figurát a sötét, ill. a világos mezőkről és megkérjük azokat, hogy lépjenek. Az egyszerűség kedvéért csak gyalogokat modellezzen! Demonstrálja a működést külön modulként fordított tesztprogrammal! A játék állását nem kell grafikusan megjeleníteni, elegendő csak karakteresen, a legegyszerűbb formában!

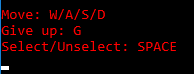
**Felhasználói leírás:**

A játékban a menüből kezdve három lehetősége van a felhasználónak: „N” betűvel az új játék, „O” betűvel a beállítások, és „X” betűvel a kikapcsolás funkciókat érhetjük el.

„O” esetén a beállítások között választhatunk játékmódot (Ember vs Ember, Ember vs Gép, Gép vs Gép), amiket rendre a „P”, „C” és „A” betűkkel választhatunk.

A játék során a „W/A/S/D” billentyűkkel irányíthatjuk a kurzort, a „SPACE”-t használva választhatunk ki és vehetjük le a jelölést egy figuráról, illetve a játék során bármikor fel lehet adni a játékot a „G” billentyűvel.

A játék végén szintén a „SPACE” lenyomásával kezdhetünk új játékot.

****

**Programkód:**

A játék kódját több különböző fájlba szedtem szét:

*C-fájlok:*

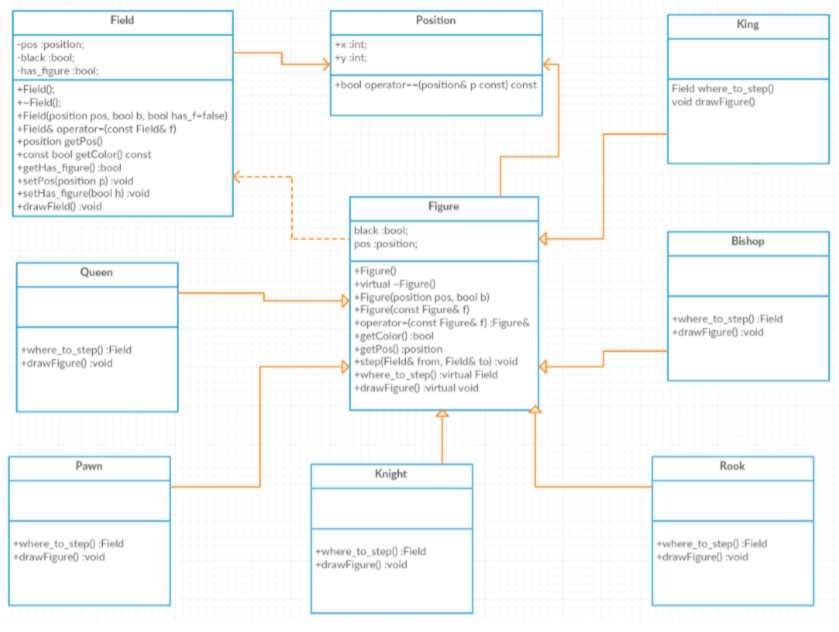
* main.cpp
* board.cpp
* field.cpp
* figure.cpp
* chess.cpp

*H-fájlok:*

* common.hpp
* board.hpp
* field.hpp
* figure.hpp
* chess.hpp

Minden c-fájlhoz (a main.c kivételével) tartozik egy header fájl is amiben az egyes c-fájlokban lévő függvényeket definiáljuk, valamint a teszteléshez és a memóriaszivárgáshoz tartozik egy “memtrace” és egy “gtest” fájl páros is.

**Egyszerűsített osztálydiagram:**



**Skeleton:**

Osztályok:

* Position
  + x, y koordinátát tárol,
  + addToPos(): egy pozíciót lehet eltolni más koordinátákra
* Field
  + látja a táblát, van színe, pozíciója, tudja milyen figura áll rajta, és hogy ki van e jelölve
  + draw(): kirajzolja a mezőt
* Figure
  + van színe, tudja melyik mezőn áll és hogy ki van-e jelölve
  + whereToStep(), whereToHit(): visszaad egy Fieldlistát ami azokat a mezőket tartalmazza ahová az adott bábu lépni/ütni tud. (csak a paraszt esetében van különbség)
  + draw(): kirajzolja a figurákat a mezőn
  + generateSteps(): kap egy irányt és egy lépésszámot és kigenerálja, hogy egy adott irányban melyik mezőre tud lépni a bábu
  + ebből származik le minden bábu illetve a FigureSet tároló is
* Board
  + két dimenziós tömbben tárol mezőket
  + checkPos(): egy pozícióról ellenőrzi, hogy a táblán belül van e
* Chess
  + a játéknak az oszálya
  + látja a táblát, van két bábukészlete illetve egy winGame változója
  + initGame(): létrehozza a figurákat és felteszi a táblára a megadott pozícióra
  + playgame(): a játék alapja, váltogatja a játékosokat és futtatja a játékot
  + oneTurn(): egy kör működéséért felelős
  + getManMove(), getAutMove(): egy lépésnek a kiválasztásáért felelős
  + step(): a lépésért felelős
  + checkAttack(): egy figuráról vizsgálja, hogy a többi támadja e

Egyéb:

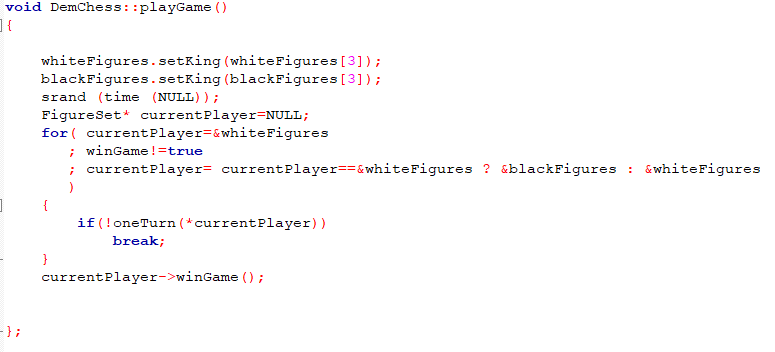
* Boardsize:
  + táblaméret=8
* Color(enum):
  + Black, White
* PlayerType(enum):
  + Manual, Auto
* FieldList(vector):
  + Mező\*-okat tárol
* FigureSet(vector):
  + Figura\*-okat tárol

**Alapvető működés:**

A játék amikor elindul, egy ciklusban kezdődik, ami a főmenüt indítja el, ami addig fut amíg az exit menüpontot nem választják. Ezen kívül van egy “New game” és egy “Options” menüpont is, amiben a játékot lehet elindítani illetve ki lehet választani, hogy gép ellen, vagy ember ellen szeretnénk játszani.

A játék elindítása után,létrejön a chess osztály egy példánya aminek a konstruktora meghívja az “initGame()” függvényt, ami az adott figurákat létrehozza, és felteszi a pályára a megadott helyre.

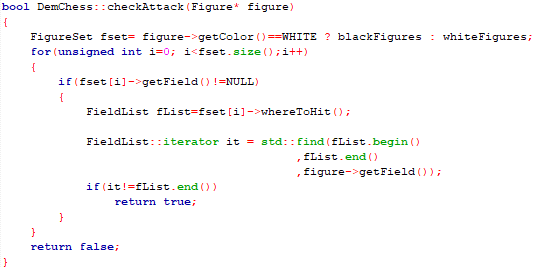
Ezek után elindul a playgame() függvény ami egy ciklusban váltogatja, hogy ki jön, és akkor áll le ha valaki nyert, vagy feladja, és a playgame() indítja el a oneturn() függvényt a cikluson belül.



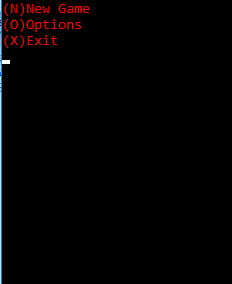
A oneturn() függvény egy kör működésért felel, megnézi, hogy gép vagy ember játszik, és aszerint hívja meg a getMove függvényt, azután ha a canmove=true (ergo van szabályszerű lépés) akkor megpróbál lépni. Ez akkor nem sikerülhet, ha a lépés által sakkba kerülne a játékos.

A getManMove() és getAutMove() függvények állítják be a “from” és “to” változókat, amiket a step() fog megkapni paraméterként. Ezen kívül, ha ember játszik, a getManMove()-on belül lehet a játékot vezérelni (és ezáltal választja ki a “from” és a “to” változókat).

Miután megvan, hogy honnan hova kell lépni, ezeket megkapja paraméterként a step(), és lép egyet. Ha ahova lépne ott sakk helyzet állna elő, akkor a lépést visszaállítja. A sakkot a checkAttack() függvény vizsgálja

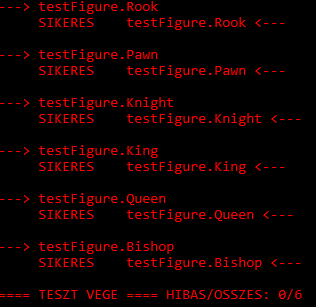
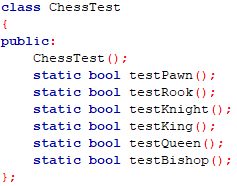
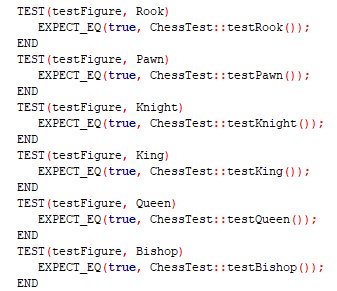


Ha már nincs olyan szabályos lépés amivel ne kerülne sakkba a játékos, vagy ha valamelyik játékos feladja, akkor a playgame() kilép a ciklusból, kiírja hogy ki nyert, és új játékot lehet kezdeni:

**Tesztelés:**

A teszteseteket egy külön test.cpp, test.hpp fájlban lehet megtalálni ami a program indításakor lefut sikeresen.



**Fejlesztési lehetőségek:**

A későbbiekben a játékot lehet fejleszteni. Például:

* Sáncolás
* En passant
* Mesterséges intelligencia a gépnek

***Smodics Roland***

***2019-05-12***